TÉCNICO LISBOA

Hacking the privacy amplification of quantum key distribution with machine learning, and countermeasures:

An argument for considering classical side channels in quantum key distribution

João Diogo Ferreira Bravo

Thesis to obtain the Master of Science Degree in

Engineering Physics

Supervisors:	Prof. Paulo Alexandre Carreira Mateus
	Prof. João Carlos Carvalho de Sá Seixas

November 2022

Dedicated to my friends and family.

Acknowledgments

I thank Instituto Superior Técnico and the Technical University of Munich for the great amount of knowledge they exposed me to and the top researchers and institutes they allowed me to be in contact with.

Moreover, I thank everyone involved in this work, specially professor Paulo Mateus for the very helpful guidance and advice throughout the research and development process and for facilitating communication with other experts in the fields involved in this work.

I also thank every single friend who supported me during this lengthy process. Thank you Supermanos for giving me multiple shoulders to lean on, picking me up when I was down, making me laugh and keeping me positive throughout the last year. Thank you Frends for your openness and invaluable friendship since high school.

I am also deeply grateful to my whole family for the constant words and actions of support. Thank you mom, dad, Inês, grandmas and everyone else.

Lastly, I thank all readers of this thesis and hope it can provide value to your personal or professional projects.

Resumo

A distribuição quântica de chaves (QKD) explora os princípios da mecânica quântica para gerar e distribuir chaves privadas usando sistemas quânticos e um canal clássico público autenticado. Apesar de incondicionalmente segura, implementações normalmente não são devido a fontes de fuga de informação inesperadas, chamadas canais laterais.

Canais laterais em QKD ou são semelhantes aos de implementações clássicas ou estão relacionados com processamento quântico. A investigação e desenvolvimento atuais têm negligenciado os primeiros.

Nesta tese, revemos sumariamente os principais avanços na análise de segurança de QKD, os principais componentes das implementações e algoritmos de pós-processamento clássico e o uso de repetidores para aumentar a distância de transmissão. Revemos integralmente os principais ataques de canal lateral a implementações de QKD e contramedidas.

Propomos um ataque de canal lateral clássico à etapa de amplificação de privacidade dum protocolo geral de QKD, usando técnicas de aprendizagem automática para analisar o seu consumo elétrico. Analisamos vários cenários simulados e conseguimos recuperar toda a chave privada múltiplas vezes. O gradient boosting foi o modelo com melhor performance em praticamente todos os cenários e recuperou toda a chave para taxas de amostragem do instrumento de medida suficientemente altas, independentemente do tamanho da matriz de hashing e do nível de ruído usados. Para modelos imperfeitos, formulamos uma estratégia que pdoe viabilizar uma procura da chave por força bruta.

Discutimos contramedidas baseadas em inserção de ruído, masking e randomização.

Este trabalho demonstra que aprendizagem automática pode ser usada para caraterizar canais laterais em QKD robustamente e executar ataques poderosos.

Palavras-chave: distribuição quântica de chaves, amplificação de privacidade, ataque de canal lateral, consumo elétrico, aprendizagem automática, gradient boosting

Abstract

Quantum key distribution (QKD) exploits the principles of quantum mechanics to generate and distribute private keys using quantum systems and an authenticated public classical channel. Although it offers information-theoretical security, its physical implementations usually do not due to unintended sources of information leakage called side channels.

Side channels in QKD are either similar to those found in classical implementations or related to the quantum processing. Current research and development have often neglected the former.

In this thesis, we briefly review the main breakthroughs in QKD security analysis, the main components of QKD implementations and algorithms for classical postprocessing and the use of repeaters for extending transmission distance. We also comprehensively review the main side-channel attacks demonstrated in QKD implementations and their countermeasures.

Moreover, we propose a classical-side-channel attack on the privacy-amplification step of a general QKD protocol based on matrix hashing, using machine-learning techniques to analyse its power-consumption leakage. We analyse multiple simulated scenarios and are often able to recover the full private key. Gradient-boosting machine was the best-performing model in virtually every scenario, recovering the full key for high-enough measuring-instrument sampling rates with any hashing-matrix size and noise level tested. In case of a non-perfect model, we devise a strategy, based on analysing confusion matrices, which can make a brute-force search for the key feasible.

We also discuss countermeasures based on noise insertion, masking and randomization techniques.

This work demonstrates that machine-learning techniques can be used to robustly characterize the leakages in a QKD implementation and generate powerful attacks.

Keywords: quantum key distribution, privacy amplification, side-channel attack, power consumption, machine learning, gradient boosting

Contents

List of Tables

List of Figures

Glossary

AdaBoost	Adaptive-boosting model. Equivalent to gradient-boosting machine with exponential loss function.
ASHA	Asynchronous successive-halving algorithm.
ASIC	Application-specific integrated circuit.
B92	Bennett protocol of 1992.
BB84	Bennett-Brassard protocol of 1984.
BBM92	Bennett-Brassard-Mermin protocol of 1992.
BS	Beam splitter.
CART	Classification-and-regression-tree algorithm.
CMOS	Complementary metal-oxide semiconductor.
CPTP	Completely-positive trace-preserving map.
CV-QKD	Continuous-variable QKD.
DEMA	Differential electromagnetic analysis.
DI-QKD	Device-independent QKD.
DPA	Differential power analysis.
DV-QKD	Discrete-variable QKD.
E91	Ekert protocol of 1991.
EB-QKD	Entanglement-based QKD.
ec	Error correction.
FEC	Forward error correction.
FPGA	Field-programmable gate array.
GBM	Gradient-boosting machine.

kNN k-nearest neighbours.

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm.

LDPC Low-density parity-check codes.

LogitBoost Gradient-boosting machine with log loss function.

MAE Mean absolute error.

MAP Maximum a-posteriori estimation.

MDI-QKD Measurement-device-independent QKD.

MLE Maximum-likelihood estimation.

MSE Mean squared error.

pa Privacy amplification.

PBS Polarization beam splitter.

pe Parameter estimation.

PM Phase modulator.

PM-QKD Prepare-and-measure QKD.

PNS Photon-number-splitting attack.

PolM Polarization modulator.

QKD Quantum key distribution.

RSA Rivest–Shamir–Adleman cryptosystem.

SAGA Stochastic average-gradient-descent algorithm.

SARG04 Scarani-Acín-Ribordy-Gisin protocol of 2004.

SEMA Simple electromagnetic analysis.

SHA Successive-halving algorithm.

SMO Sequential minimal optimization.

SPA Simple power analysis.

SPD Single-photon detector.

SVM Support-vector machine.

VC Vapnik-Chervonenkis theory.

Chapter 1

Introduction

We begin this work by introducing the fields of cryptography and quantum key distribution and motivating the need for this research in section 1.1. In section 1.2, we state the objectives of our research and, in section 1.3, we outline the contents of the different chapters of this thesis.

1.1 Motivation

When we want to share a secret message over an untrusted channel, it is essential that we encrypt that message if we want to ensure it does not fall into the wrong hands. Cryptography studies protocols for secure communication in the presence of an adversarial party. These usually include the encryption of messages with a secret key.

Two well-known classes of such protocols, also known as cryptosystems, are symmetric- or private-key cryptosystems [1], which use identical private keys for both encryption and decryption of messages, and asymmetric- or public-key cryptosystems [2], which use a public key to encrypt messages and a private key to decrypt them. The former protocols are difficult to implement alone in a secure way due to the need to keep keys private during the key-exchange process. By contrast, public-key protocols have achieved widespread use since their proposal in 1976, with the Rivest-Shamir-Adleman (RSA) cryptosystem [3] being a prime example.

The security of key distribution based on public keys and classical channels relies on the unproven assumption that eavesdroppers lack the computational resources needed to solve a specific mathematical problem [1], namely the inversion of one-way functions. Thus, public-key protocols are said to be computationally secure. However, scientific and technological advances are progressively weakening this form of security. In fact, standard public-key algorithms, such as RSA, will be broken by a sufficiently-powerful quantum computer in the future [4]. This lack of long-term security has generated the need for new protocols with a stronger notion of security.

In quantum cryptography, we exploit the principles of quantum mechanics to perform cryptographic tasks, such as key exchange [4]. This is usually called quantum key distribution (QKD) and consists in generating and distributing a private key using quantum systems and an authenticated public classical channel, i.e., a classical channel where an eavesdropper is not able to impersonate one of the two parties but can still eavesdrop on them. This key can then be used in a symmetric protocol, for instance. QKD offers information-theoretical security [5], also known as unconditional security. This means it can be proven secure with information-theory methods without imposing any conditions on the resources available to an eavesdropper. Thus, in theory, QKD protocols cannot be broken even by an eavesdropper with unlimited classical and quantum computing power.

Despite its information-theoretical security, QKD faces a few practicality challenges. Some of the major ones concern its limitations in transmission distance and key generation rate [6]. These limitations come from the instability of the quantum systems used to encode the transmitted information. These systems are prone to decoherence by interacting with the quantum channel over which they are transmitted, which results in information loss. At present, few basic QKD implementations are able to perform key exchange over more than 500 km of ultra-low-loss optical fiber [7].

A solution to this challenge is the use of repeaters, also known as nodes or relays, along the communication channels. These repeaters work either by executing key exchanges between consecutive repeaters and encrypting the final private key with those keys [8] or by correcting the decoherence in the transmitted quantum systems through a process called entanglement distillation [9]. The latter type of repeaters is still an embryonic technology [5].

Another major challenge in the practical use of QKD is due to unintended sources of information leakage in physical implementations [5], since these leakages are often highly correlated with the generated key. Such leakage sources are commonly called side channels. An eavesdropper can interact with and exploit physical side channels rather than the abstract protocol to obtain information about the generated key without being detected.

We will divide side channels in QKD into two categories according to their origin. We will refer to side channels related to the quantum processing of a QKD protocol as quantum side channels. On the other hand, side channels such as power consumption [10], emanated electromagnetic radiation [11] and even sound produced by implementation devices [12] are also found in classical cryptographic implementations. Thus, we will refer to them as classical side channels.

There has been substantial research on quantum-side-channel attacks [5], with multiple security patches proposed for each attack. Besides specific security patches, new classes of protocols have been proposed which claim to make side-channel attacks fully or partially obsolete. The most-well-known classes are measurement-device-independent (MDI) QKD [13] and device-independent (DI) QKD [14]. The former claims to remove all side channels from detection components, while the latter claims to remove all side channels from all implementation components. However, we argue that DI-QKD protocols may still be exposed to classical side channels since these do not disturb the transmitted quantum systems. Moreover, MDI-QKD protocols may still be exposed to both classical and quantum side channels in source components.

Unfortunately, classical-side-channel attacks are often not taken into account in QKD research and development. In fact, only very recently there have been a few proposals of such attacks on QKD implementations, targeting side channels such as timing information [15], power consumption [16, 17], emanated electromagnetic radiation [18, 19] and cache state [20] of specific implementation devices. However, most devices used in a QKD implementation may be vulnerable to classical-side-channel attacks. Moreover, QKD networks with multiple remotely-located repeaters can be particularly exposed to such attacks since constant surveillance of all repeaters can be difficult. Therefore, we believe greater focus should be given to research on classical-side-channel attacks on QKD implementations to maximize their practical security.

1.2 Objectives

The aim of our thesis is to contribute to the research on classical-side-channel attacks on QKD implementations, while also reviewing some of the main theoretical and experimental progress in the field of QKD.

Our main objectives are to:

1. briefly review some of the major theoretical and experimental advances in QKD and its most-used current techniques;
2. comprehensively review the main quantum- and classical-side-channel attacks demonstrated in QKD implementations and their respective countermeasures;
3. propose and analyse a new classical-side-channel attack on the privacy-amplification step of a general QKD protocol based on matrix hashing, using its power-consumption leakage. For this, we will simulate the power-consumption leakage of a specific implementation for privacy amplification in multiple attack scenarios. Then, we will analyse all leakages with machine-learning techniques;
4. provide different approaches to analyse leakages as a supervised-learning problem and assess the best approach and machine-learning model to perform our proposed attack. The aim is to study the exploitability of classical side channels in QKD implementations with the aid of machine-learning techniques;
5. in case of a successful attack, propose countermeasures to prevent similar side-channel attacks and discuss their efficiency.

1.3 Thesis outline

The remainder of this thesis is organized as follows: in chapter 2, we introduce the basics of QKD, reviewing some of its theoretical and experimental progress, current techniques and side-channel attacks. We also present the theory behind the machine-learning techniques used in a reasonably-self-contained way, focusing on supervised learning and its general workflow. We leave in-depth mathematical descriptions of the models used to appendix A; in chapter 3, we present our proposed attack on the privacy-amplification step of a general QKD protocol based on matrix hashing, using its power-consumption leakage. We also detail the implementation considered, how we simulated the power-consumption leakages, the machine-learning-based approaches used and the considered attack scenarios; in chapter 4, we present our results, analysing both the simulated power-consumption leakages and the results of the various approaches used for each attack scenario. We then compare and choose the best approach depending on the attack scenario and discuss possible countermeasures to prevent similar side-channel attacks. A summary of all results can be found in appendix B; finally, in chapter 5, we describe our main achievements, discuss further considerations and suggest possible future research directions.

Chapter 2

Background

In this chapter, we introduce the topic of QKD, reviewing some of its theoretical and experimental progress, current techniques and side-channel attacks. We also present the theory behind the machine-learning techniques used in a mostly-self-contained way, focusing on supervised learning and its general workflow.

In section 2.1, we present the mathematical formalism necessary to understand the following exposition of QKD. In section 2.2, we delve into QKD, focusing on discrete-variable protocols. We describe entanglement-based and prepare-and-measure protocols and provide examples in sections 2.2.1 and 2.2.2, respectively. In section 2.2.3, we review the common assumptions of QKD security proofs along with the main breakthroughs in QKD security analysis. In section 2.2.4, we describe the main components of QKD implementations, the most common discrete-variable implementations, the main algorithms for classical postprocessing and the use of repeaters for extending the transmission distance of QKD implementations. In section 2.3, we present the concepts of side channel and side-channel attack, distinguishing classical side channels from quantum side channels. In section 2.3.1, we review some of the main side-channel attacks demonstrated in QKD implementations and proposed countermeasures.

In section 2.4, we delve into machine learning, focusing on supervised learning. We describe the data used in supervised learning and how to use it for model training in section 2.4.1. In section 2.4.2, we describe the data-preprocessing techniques we used. In section 2.4.3, we provide a brief description of the models used in our analysis. More in-depth mathematical descriptions of those models are provided in appendix A. In section 2.4.4, we introduce the task of hyperparameter optimization, along with the main techniques used for training, testing and selecting models, including lower-fidelity approximation methods. In section 2.4.5, we present the model-evaluation metrics we used in our analysis and, in figure 2.8, we summarize the workflow of a supervised-learning analysis.

2.1 Mathematical formalism

Here, we define crucial mathematical objects necessary to understand our exposition regarding quantum key distribution in later sections.

2.1.1 Quantum bits

Analogous to a bit in classical information, a quantum bit, commonly known as qubit [4], is the basic unit used in quantum information. It describes a two-level quantum system, such as the spin of electrons or the polarization of photons.

A qubit in a pure state may be described as a linear operator on a Hilbert space, known as state vector [4]. Using Dirac notation, this operator is represented as

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,, \quad a, b \in \mathbb{C}, \quad |a|^2 + |b|^2 = 1. \tag{2.1}$$

Due to noise and decoherence, a qubit's state can become partially unknown. We say it enters a mixed state, where it is described by a statistical mixture of pure states. In this case, it can be represented by a linear operator on a Hilbert space known as density operator [4], which satisfies

$$\rho = \sum_i p_i\,|\psi_i\rangle\langle\psi_i|\,, \quad tr[\rho] = 1, \quad \rho \geq 0, \quad \rho = \rho^\dagger, \tag{2.2}$$

where p_i is the probability of the qubit being in state $|\psi_i\rangle$, tr is the trace operator, $\rho \geq 0$ indicates ρ is positive semi-definite and $\rho^\dagger$ is the conjugate transpose of ρ. It is also possible to represent pure states as density operators. For instance, the state in equation (2.1) can be represented as

$$\rho = |\psi\rangle\langle\psi| = \begin{bmatrix} |a|^2 & ab^* \\ a^*b & |b|^2 \end{bmatrix}, \tag{2.3}$$

where a^* and b^* are the complex conjugates of a and b, respectively.

If two parties A and B share qubits, the total state of their systems is referred to as a bipartite quantum state [4]. It is described by an operator on the tensor product of their respective Hilbert spaces $\mathcal{H}_A \otimes \mathcal{H}_B$ as

$$\rho_{AB} = \sum_{i,j} p_{ij}\rho_A^i \otimes \rho_B^j, \tag{2.4}$$

where $\{\rho_A^i\}$ and $\{\rho_A^j\}$ are the bases of density operators in $\mathcal{H}_A$ and $\mathcal{H}_B$, respectively.

2.1.2 Quantum channels and measurements

A completely-positive trace-preserving (CPTP) map $\mathcal{M} : \mathcal{H}_A \to \mathcal{H}_B$ satisfies the conditions [21]

$$\rho' \geq 0 \implies (\mathbb{I}_n \otimes \mathcal{M})(\rho') \geq 0, \quad \forall\, \rho' \in \mathbb{C}^{n\times n} \otimes \mathcal{H}_A, \quad \forall\, n, \tag{2.5}$$

$$tr\,[\mathcal{M}(\rho)] = tr[\rho], \quad \forall\, \rho \in \mathcal{H}_A, \tag{2.6}$$

where $\mathbb{I}_n$ is the identity map on $\mathbb{C}^{n\times n}$. The first condition ensures complete positivity, while the second guarantees trace preservation.

A CPTP map $\mathcal{N} : \mathcal{H}_A \to \mathcal{H}_B$ on the set of density operators is called a quantum channel and can be used to model the transmission of quantum information [21].

A generalized measurement of a quantum system can be described by a set of linear operators $\{M^x\}_{x\in X}$ on a Hilbert space [4], where X is the set of possible outcomes. The set of operators $\{E^x\}_{x\in X}$, where $E^x = (M^x)^\dagger M^x$, is known as a positive operator-valued measure (POVM) and the operators M^x are known as Kraus operators. These satisfy

$$E^x = (M^x)^\dagger M^x \geq 0, \quad \forall x \in X, \tag{2.7}$$

$$\sum_{x\in X} E^x = \sum_{x\in X} (M^x)^\dagger M^x = \mathbb{I}. \tag{2.8}$$

2.1.3 Quantum metrics

The ∞-norm, also known as operator norm, for a linear operator $M : \mathcal{H}_A \to \mathcal{H}_B$ is defined as [21]

$$\|M\|_\infty = \sup_{\substack{\rho\in\mathcal{H}_A \\ \rho\neq 0}} \frac{|M\rho|}{|\rho|}, \tag{2.9}$$

where $|\rho| = \sqrt{\rho^\dagger \rho}$. If $\mathcal{H}_A$ and $\mathcal{H}_B$ have finite dimensions, $\|M\|_\infty$ is simply the largest singular value of M, i.e., the largest eigenvalue of $|M|$.

We define the trace distance between density operators ρ and σ as [21]

$$\|\rho - \sigma\|_{tr} = \frac{1}{2}\|\rho - \sigma\|_1 = \sup_P tr[P(\rho - \sigma)] \in [0, 1], \tag{2.10}$$

where $\|\rho\|_1 = tr[\,|\rho|\,]$ and P ranges over all projectors, i.e., positive semi-definite operators with eigenvalues in $\{0, 1\}$. The trace distance measures how distinguishable two states are. In particular, the maximum probability of distinguishing two states with trace distance ϵ with a single measurement is $\frac{1}{2}(1 + \epsilon)$.

We define the diamond distance between two CPTP maps $\mathcal{M}$ and $\mathcal{F}$ as [22]

$$\|\mathcal{M} - \mathcal{F}\|_\diamond = \sup_{\rho_{AE}} \|\mathcal{M}(\rho_{AE}) - \mathcal{F}(\rho_{AE})\|_{tr} \in [0, 1], \tag{2.11}$$

where ρ_{AE} ranges over all states on $\mathcal{H}_A$ and its extensions to an auxiliary space $\mathcal{H}_E$. This distance may also be called completely-bounded trace distance and measures how distinguishable two CPTP maps are. Let us assume we have two CPTP maps with diamond distance β. Then, analogously to the trace distance, the maximum probability of distinguishing them by applying each on a joint state on the input space and an auxiliary space and then measuring the resulting state is $\frac{1}{2}(1 + \beta)$.

2.1.4 Hash functions

A hash function is any function $H : X \to K$ such that $|X| \geq |K|$ [23], where $|X|$ is the cardinality of X. The objects in K are called hashes.

Let $\mathcal{H} = \{h : X \to K\}$ be a family of hash functions. $\mathcal{H}$ is universal$_2$ if [23]

$$x \neq x' \implies P\big(H(x) = H(x')\big) \leq \frac{1}{|K|}, \quad \forall\, x, x' \in X, \tag{2.12}$$

when H is chosen uniformly at random from $\mathcal{H}$.

We can further generalize this notion to a δ-almost universal$_2$ family of hash functions [24]. Such a family satisfies

$$x \neq x' \implies P\big(H(x) = H(x')\big) \leq \delta, \quad \forall\, x, x' \in X, \tag{2.13}$$

when H is also chosen uniformly at random.

2.2 Quantum key distribution

As discussed in section 1.1, QKD is a quantum-cryptographic task which allows two parties to generate a shared private key using quantum systems and an authenticated public classical channel. In cryptography, these communicating parties are usually designated by Alice (A) and Bob (B), while a possible eavesdropper is called Eve (E) [4]. At the end of a QKD protocol, the resulting private key can be used to encrypt messages over an untrusted classical channel, i.e., a classical channel which may be subject to eavesdropping and does not require authentication.

We note that, for QKD to be feasible, the two parties must initially share a small private, or weakly-correlated and partially-private [25], key to authenticate the public channel and verify each other's identities. After the first communication round, a part of the generated key can be used for reauthentication purposes [26]. Due to this initial constraint, QKD could also be called quantum key expansion.

QKD offers information-theoretical security [5]. This is done by exploiting the principles of quantum mechanics. In particular, all QKD protocols use the fact that measuring a quantum system perturbs its state to detect the presence of an eavesdropping third party. The way in which this is achieved depends on the category of protocols considered. The most well-known QKD protocols can be divided into two categories, depending on which quantum property they exploit [5]: entanglement-based (EB) protocols and prepare-and-measure (PM) protocols. These will be described in sections 2.2.1 and 2.2.2, respectively.

These two approaches can be further divided into two families of protocols: discrete-variable (DV) protocols and continuous-variable (CV) protocols [5]. DV-QKD uses finite-dimensional quantum systems, such as qubits. CV-QKD uses infinite-dimensional quantum systems, such as bosonic modes of the electromagnetic field, and these protocols were initially proposed as a means of overcoming limitations in the physical implementations of DV-QKD protocols. We will focus our description mainly on DV-QKD for simplicity.

The first QKD protocol was a discrete-variable prepare-and-measure protocol known as BB84 and proposed in 1984 by Bennett and Brassard [27], which makes QKD the first quantum-information technology.

2.2.1 Entanglement-based QKD

EB-QKD protocols take advantage of quantum entanglement [5]. This property describes groups of smaller subsystems, such as particles, in which a subsystem's state cannot be represented independently of the states of the other subsystems, e.g., an entangled state $|\psi\rangle_{AB}$ of subsystems A and B cannot be represented as $|\psi\rangle_A \otimes |\psi\rangle_B$, where $|\psi\rangle_A$ is the representation of the state of system A independently of the state of system B, and vice-versa. Therefore, we can only make assertions about the system as a whole, even if the subsystems are physically separated by large distances. Moreover, performing a measurement in one subsystem affects the others in such a way that, if an eavesdropper, Eve, intercepted one subsystem, she would alter the total system. Thus, this would reveal her presence and even the amount of information she obtained.

From a mathematical perspective, an EB-QKD protocol is a CPTP map, composed of local operations and classical communication, which takes a bipartite quantum state ρ_{AB} and either aborts due to interference or outputs two classical binary strings $\boldsymbol{k}_A$ and $\boldsymbol{k}_B$ [22].

A notable example of EB-QKD is the Bennett-Brassard-Mermin (BBM92) protocol [28]. A major family of DV-EB-QKD protocols, based on the BBM92 protocol, can be parametrized by the following parameters [22]:

- Block length, $m \in \mathbb{N}$, the number of individual quantum systems shared between Alice and Bob. It defines the maximum length of the bit strings used in the protocol;
- Parameter estimation tuple, $pe = \{k, \delta\}$, where $k \in \mathbb{N}$, $k \leq m$ is the number of quantum systems used for parameter estimation, and $\delta \in (0, \frac{1}{2})$ is the tolerated error rate. We will use $n = m - k$ to denote the number of systems used for key generation;
- Error correcting tuple, $ec = \{r, \text{synd}, \text{corr}, t, \mathcal{H}_{ec}\}$, where $r \in \mathbb{N}$ is the bit length of the error-correction syndrome, i.e. the bit string which is the description of which qubits are corrupted; synd : $\{0,1\}^n \to \{0,1\}^r$ is the function which outputs the syndrome; corr : $\{0,1\}^n \times \{0,1\}^r \to \{0,1\}^n$ is the function which uses the syndrome to output a corrected string; $t \in \mathbb{N}$ is the bit length of the hash used for verifying the corrected string; and $\mathcal{H}_{ec} = \{h_{ec} : \{0,1\}^n \to \{0,1\}^t\}$ is a universal$_2$ family of hash functions;
- Privacy amplification tuple, $pa = \{l, \mathcal{H}_{pa}\}$, where $l \in \mathbb{N}$, $l \leq n$ is the bit length of the final secret key, which is extracted from one block of measurement data, and $\mathcal{H}_{pa} = \left\{h_{pa} : \{0,1\}^n \to \{0,1\}^l\right\}$ is a universal$_2$ family of hash functions.

The ratio of the key length to the block length $\frac{l}{m}$ is a performance indicator designated by key generation rate.

This family of EB-QKD protocols is defined by the steps:

1. **Input**: Alice and Bob generate a bipartite entangled quantum state ρ_{AB} with a possibly-untrusted source, where A and B are each composed of m joint quantum systems of finite dimension. They also define two orthogonal bases for measuring the quantum systems and assign the bits 0 or 1 to the two possible measurement outcomes in each basis;

2. **Randomization**: Alice and Bob generate and share uniformly-random seeds over the authenticated public channel. The following random seeds are generated:

    - Random string $\boldsymbol{\phi} \in \{0,1\}^m$: defines the sequence of bases for measuring the m quantum systems of each party;
    - Random subset $\Pi \in \{\pi \subset \{1, ..., m\} : |\pi| = k\}$: defines the k quantum systems whose measurement bits will be used for parameter estimation. The remaining n quantum systems will be used to generate a key;
    - Random hash function $H_{ec} \in \mathcal{H}_{ec}$: hash function to use in the error-correction step;
    - Random hash function $H_{pa} \in \mathcal{H}_{pa}$: hash function to use in the privacy-amplification step;

3. **Measurement**: Alice and Bob measure their m quantum systems according to bases defined by $\boldsymbol{\phi}$. Then, they each store their k outcome bits corresponding to the systems defined by Π in strings $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, respectively, which will be used for parameter estimation, and the remaining n outcome bits in strings $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$, respectively, for error correction;

4. **Parameter estimation**: Alice sends $\boldsymbol{v}_A$ to Bob over the authenticated public channel. Bob then compares it with $\boldsymbol{v}_B$ and assesses the fraction of errors, given by

$$\Delta(\boldsymbol{v}_A, \boldsymbol{v}_B) = \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{1}(v_{A,i} \neq v_{B,i}), \tag{2.14}$$

    where $v_{A,i}$ and $v_{A,i}$ are the bit i of strings $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, respectively, and $\mathbb{1}$ is the 0-1 indicator function which outputs 1 when its argument is true and 0 when it is false. If $\Delta(\boldsymbol{v}_A, \boldsymbol{v}_B) > \delta$, Bob sets the interference flag $F^{pe} = 1$ and aborts the protocol. Otherwise, he sets $F^{pe} = 0$ and they proceed to the next step;

5. **Error correction**: Alice sends the error syndrome $\boldsymbol{z} = \text{synd}(\boldsymbol{x}_A)$ to Bob over the authenticated public channel. Then, Bob computes the corrected string $\hat{\boldsymbol{x}}_B = \text{corr}(\boldsymbol{x}_B, \boldsymbol{z})$. In order to verify that the error correction succeeded with high probability, Alice computes the hash $H_{ec}(\boldsymbol{x}_A)$ and sends it to Bob over the public channel. Bob computes $H_{ec}(\hat{\boldsymbol{x}}_B)$ and compares it with $H_{ec}(\boldsymbol{x}_A)$. If the hashes differ, he sets the interference flag $F^{ec} = 1$ and aborts. Otherwise, he sets $F^{ec} = 0$ and they proceed to the next step;

6. **Privacy amplification**: Alice and Bob compute the secret hashes $\boldsymbol{k}_A = H_{pa}(\boldsymbol{x}_A)$ and $\boldsymbol{k}_B = H_{pa}(\hat{\boldsymbol{x}}_B)$, respectively, both of length l. These hashes become the final secret key.

Besides the BBM92 protocol, which uses two orthogonal measurement bases, another well-known example of DV-EB-QKD is the E91 protocol [29], which uses three non-orthogonal bases and relies on a generalization of Bell's theorem, known as the CHSH inequality [30], to detect eavesdropping.

2.2.2 Prepare-and-measure QKD

The second main category are PM-QKD protocols. These protocols take advantage of the probabilistic nature of quantum systems, usually referred to as quantum uncertainty or quantum indeterminacy, to detect any eavesdropping [5]. Since eavesdropping necessarily involves measuring the quantum systems, it changes their states and, consequently, the probability distribution of the following measurement outcomes of their observables, which can reveal Eve's presence along with the amount of disclosed information.

In PM-QKD protocols, Alice and Bob do not start with an entangled state. Instead, they have access to an quantum channel, which may be untrusted. Since maintaining a shared entangled state at long distances is very challenging in practice, EB-QKD protocols are generally considered idealized versions of PM-QKD protocols [22]. The latter are often called realistic protocols.

Mathematically, a PM-QKD protocol is identical to an EB-QKD protocol, except it takes a quantum channel as input instead of a bipartite quantum state [22].

A notable family of DV-PM-QKD protocols, based on the BB84 protocol, can be parametrized by the same parameters of the above EB-QKD-protocol family, with the addition of [22]:

- Number of states prepared and sent through the quantum channel by Alice, $M \in \mathbb{N}$, $M \geq m$.

The secret-key generation rate in these protocols is now $\frac{l}{M}$, which is $\frac{m}{M}$ smaller than in the equivalent EB-QKD scheme. This illustrates the necessary trade-off between ease of implementation and degree of security.

This family of PM-QKD protocols is defined by the steps:

1. **Input**: Alice and Bob setup a quantum channel $\mathcal{N}_{A\to B}$ capable of transmitting M quantum systems from Alice to Bob. Additionally, they define two orthogonal bases for measuring the quantum systems and assign the bits 0 or 1 to the two possible measurement outcomes in each basis, as in the EB-QKD case;

2. **Randomization**: Alice generates the following uniformly-random seeds:

    - Random string $\boldsymbol{\phi}_A \in \{0,1\}^M$: defines the sequence of bases in which to prepare her M quantum systems;
    - Random string $\boldsymbol{r} \in \{0,1\}^M$: defines the sequence of bits to encode in her M quantum systems, by preparing them in the corresponding states of the bases defined by $\boldsymbol{\phi}_A$;
    - Random subset $\Pi \in \{\pi \subset \{1,...,m\} : |\pi| = k\}$: as in the EB-QKD case, defines the k quantum systems to use for parameter estimation. The remaining $n = m - k$ quantum systems will be used for key generation;
    - Random hash function $H_{ec} \in \mathcal{H}_{ec}$: hash function to use in the error-correction step;
    - Random hash function $H_{pa} \in \mathcal{H}_{pa}$: hash function to use in the privacy-amplification step.

    Bob generates the uniformly-random seed:

- Random string $\boldsymbol{\phi}_B \in \{0,1\}^M$: defines the sequence of bases for measuring the M systems sent by Alice.

3. **State preparation**: Alice prepares a quantum state $\rho_A^{\boldsymbol{\phi}_A,\boldsymbol{r}}$, composed of M joint quantum systems of finite dimension, by encoding $\boldsymbol{r}$ in the bases defined by $\boldsymbol{\phi}_A$;

4. **State distribution**: Alice sends $\rho_A^{\boldsymbol{\phi}_A,\boldsymbol{r}}$ to Bob over the quantum channel. In practice, Alice can send the systems individually, using the quantum channel M times;

5. **Measurement**: Bob measures the M quantum systems in the bases defined by $\boldsymbol{\phi}_B$. Then, he stores the outcome bits in string $\boldsymbol{u}$, denoting the inconclusive outcomes by $\varnothing$. He also registers the subset $\Omega = \{i \in \{0,...,M-1\} : u_i \neq \varnothing\}$, which indicates which measurement outcomes were conclusive. When using photons to encode the transmitted qubits, inconclusive measurement outcomes could be due to photon loss or dark counts, for instance. Photon loss occurs when none of Bob's detectors registers a photon, and dark counts occur when more than one of his detectors register a photon;

6. **Randomness distribution**: Bob sends $\boldsymbol{\phi}_B$ and Ω to Alice over the authenticated public channel, and Alice sends Π, H_{ec} and H_{pa} to Bob.

7. **Sifting**: Alice computes the set $\Sigma = \{i \in \Omega : \phi_{A,i} = \phi_{B,i}\}$, which indicates which conclusive measurements used the same basis. If $|\Sigma| < m$, she sets the flag $F^{si} = 1$ and aborts the protocol. Otherwise, she sets $F^{si} = 0$, randomly chooses a subset $\Lambda \subseteq \Sigma$ with cardinality m, and sends Λ to Bob over the authenticated public channel. Then, Alice and Bob respectively generate strings $\boldsymbol{r}'$ and $\boldsymbol{u}'$ by discarding the bits of $\boldsymbol{r}$ and $\boldsymbol{u}$ which do not correspond to the measurements defined by Λ. As in the EB-QKD case, the k bits of $\boldsymbol{r}'$ and $\boldsymbol{u}'$ corresponding to the measurements on the systems defined by Π are stored in strings $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$, respectively, to be used for parameter estimation. The remaining n bits are stored in strings $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$, respectively, for error correction;

8. **Parameter estimation**: equal to the EB-QKD case;

9. **Error correction**: equal to the EB-QKD case;

10. **Privacy amplification**: equal to the EB-QKD case.

This family of PM-QKD protocols can be reduced to the family of EB-QKD protocols presented in section 2.2.1, as shown in [22]. This implies that, if one of the families of protocols is proven to be secure, the other one must also be.

Some well-known examples of DV-PM-QKD are:

- the above-mentioned BB84 protocol, which uses two orthogonal bases;

- the B92 protocol [31], which is a simplified version of the BB84 which uses only two non-orthogonal quantum states;

- the six-state protocol [32], which is another variant of the BB84 which uses six quantum states and three orthogonal bases. This extra basis choice allows for more noise in the quantum channel and increases the probability of detecting an eavesdropper;
- the SARG04 protocol [33], which only differs from the BB84 in the sifting step, but is more robust against photon-number-splitting (PNS) attacks, described in section 2.3.1.

It is important to note that most of these protocols have also been adapted to EB-QKD versions [5].

2.2.3 Security of QKD

All the above-mentioned protocols have been proven to be information-theoretically secure [5]. In fact, multiple versions of security analyses have been proposed for various QKD protocols. Here, we describe what we consider the main breakthroughs in QKD security analysis.

Common assumptions

While information-theoretical security implies that no condition is imposed on the resources available to an eavesdropper, different assumptions must be made about the implementations of Alice and Bob. Some of the most common assumptions of the proofs mentioned below are:

- Alice and Bob have sealed spatially-separated laboratories so that they control exactly what information they release from their laboratory and we can model each quantum system shared between them as a tensor product of the their respective local Hilbert spaces;
- Alice and Bob have access to ideal uniformly-random seeds produced by a trusted quantum random-number generator, for instance;
- Alice and Bob are able to securely authenticate their public classical communication channel;
- for DV-QKD protocols, Alice and Bob share finite-dimensional quantum systems;
- for CV-QKD protocols, Alice and Bob share infinite-dimensional quantum systems;
- The states of the quantum systems leak no information to Eve about the basis choice in which they were prepared;
- for EB-QKD protocols, Alice and Bob's measurements commute, i.e., the order in which they measure their quantum systems does not affect the resulting outcome distribution. In other words, their measurement devices have no memory effects. This is a common assumption of QKD with trusted measurement devices;
- for PM-QKD protocols, instead of commuting measurements, it is usually assumed that the state of each quantum system does not depend on the preparation of the other quantum systems;

- for protocols such as the ones of the EB-QKD family defined in section 2.2.1, Alice's measurements are assumed to either be perfectly complementary or sufficiently complementary, i.e., their average overlap $\overline{c}$ must satisfy

$$\begin{aligned} \overline{c}(m,n) &= \max_{\Pi \in \{\pi \subset \{1,\ldots,m\} : |\pi| = n\}} \left(\prod_{i \in \Pi} c_i \right)^{\frac{1}{n}} < 1, \\ \text{with} \quad c_i &= \max_{x,y \in \{0,1\}} \left\| M_{A_i}^{0,x} \left(M_{A_i}^{1,y} \right)^\dagger \right\|_\infty^2, \end{aligned} \tag{2.15}$$

where Alice's i-th measurement with basis ϕ_i is modeled by the binary generalized measurement $\left\{ M_{A_i}^{\phi_i,x} \right\}_{x \in \{0,1\}}$, with x denoting the outcome. For instance, ideal measurements in the computational and the Hadamard bases are perfectly complementary, with $c_i = \frac{1}{2}$, $\forall\, i \in \{1, ..., m\}$. The average overlap in an implementation can be measured experimentally even if the Kraus operators are not known [34];

- for protocols such as the ones of the PM-QKD family defined in section 2.2.2, analogously to the assumption of measurement complementarity, it is usually assumed that the prepared states are either perfectly complementary or sufficiently complementary, i.e.,

$$\begin{aligned} \overline{c}' &= \max_{i \in \{1,\ldots,M\}} c_i' < 1 \\ \text{with} \quad c_i' &= \max_{x,y \in \{0,1\}} \left\| \sqrt{\rho_{A_i}^{0,x}}\, X^{-1} \sqrt{\rho_{A_i}^{1,y}} \right\|_\infty^2, \\ X &= \sum_{x \in \{0,1\}} \rho_{A_i}^{0,x} = \sum_{y \in \{0,1\}} \rho_{A_i}^{1,y}, \end{aligned} \tag{2.16}$$

where $\rho_{A_i}^{\phi_i,x}$ represents the state of the i-th quantum system when prepared in the basis ϕ_i and encoding the bit x. Perfectly-complementary states have $\overline{c}' = \frac{1}{2}$.

Main breakthroughs

One of the first rigorous proofs of the information-theoretical security of QKD was given in [35], where it was considered the asymptotic limit of infinite exchanged quantum systems over arbitrarily-long distances and unbounded classical computing power. The proof considered a DV-EB-QKD protocol which used fault-tolerant quantum computers for parameter estimation and error correction. It also considered noisy state-preparation and measurement devices, in which the initial bipartite quantum state was shared over a noisy untrusted quantum channel. Then, it reduced the noisy quantum protocol to a noiseless classical protocol, which was tackled with classical probability theory to prove that, whenever Eve's attack had a non-negligible probability of remaining unnoticed, the final-key information she obtained decreased exponentially with the increase in the key length. Nevertheless, this proof has the drawback of requiring fault-tolerant quantum computers, a technology which is not yet available.

Later, [36] rigorously proved the security of the BB84 protocol by first proving the security of a DV-EB-QKD protocol using the same methods as [35] and then showing that protocol could be reduced to the BB84 protocol. Shortly after, [37] presented a security proof for the BB84 protocol which also considered noisy quantum channel and measurement devices. This proof is based on an earlier version [38] which some consider the first rigorous proof of the information-theoretical security of QKD. Both [36, 37] considered the same limit of infinite exchanged quantum systems and unbounded classical computing power as [35]. However, both assumed ideal single-photon sources as state-preparation devices. Therefore, their proof does not hold in case Alice sends two or more identical photons at a time.

[39] addressed this issue by proving the security of more practical implementations of the BB84 protocol against individual attacks, i.e., attacks based on interacting with and measuring each quantum system independently. These implementations used realistic sources, such as parametric-down-conversion sources or multi-photon sources which emit weak coherent pulses. However, information-theoretical security implies resistance against the most general attacks, namely coherent attacks, in which Eve interacts with and measures all quantum systems jointly based on knowledge from all the classical-postprocessing communication [5]. Afterwards, [40] combined the model in [39] with the proof of [37] to show that such realistic implementations can also have information-theoretical security. Nevertheless, it has also been shown that the security of similar protocols, which use multi-photon sources, is always threatened if the loss in the quantum channel is sufficiently high [41]. Based on works such as [39, 40], [42] proposed a general framework for analysing QKD protocols with realistic devices. This framework consists in characterizing the imperfections in physical devices by comparing their output with the one expected from ideal devices. Those imperfections are taken into account in the security analysis by tagging the outputs which differ from the ideal case and treating them differently than the rest. Thus, this framework describes how to extract secure bits from a mix of reliable and unreliable bits. It can be applied to protocols with imperfect sources, quantum channels and measurement devices. Some examples of these imperfections are photon sources which occasionally emit weak coherent pulses, weak basis-dependent biases in both sources and measurement devices, and misalignment of source and measurement devices.

The early security proofs in [35–37] use techniques related to quantum error correction, which, at first, couple the initial error correction and the privacy amplification steps in the security analysis. By noting that the initial error correction is not necessarily related to quantum laws in the security analysis, i.e., Alice and Bob may generate insecure identical keys by simply sharing classical states, [43] proposed a security-analysis framework which uses an entropic form of Heisenberg's uncertainty principle, known as generalized entropic uncertainty relations [44]. This framework does not require us to use quantum-error-correcting codes. Thus, it decouples the error correction step from the privacy amplification step in the security analysis from the beginning, simplifying the proofs. It also applies to implementations with imperfect devices. Very recently, an equivalence between the approach with quantum-error-correcting codes and the entropic approach was established [45].

Based on the entropic approach, [46] proposed a simplified security-analysis framework which relies on the complementarity of the measurement bases. This framework also does not require the use of quantum-error-correcting codes. Its key idea is to consider the errors produced by imperfect devices, i.e, bit differences between Alice and Bob even when they use the same measurement bases, and treat them separately, depending on the basis they use. It can be applied to the BB84 protocol and any variant which uses complimentary bases, such as the six-state protocol.

All the above works limited their analyses to to the case of Alice and Bob having infinite resources. In practice, their resources are necessarily finite, i.e., they exchange a finite number of quantum systems and have a limited amount of classical computing power available, which leads to secret keys with finite length. In this regime of finite resources, perfect security is no longer achievable and a notion of approximate security is needed. [47, 48] accomplished this by bringing the concept of universal composability [49] from classical cryptography to the QKD setting. A cryptographic protocol is said to have universally-composable security if it remains secure in any arbitrary context, even when used within a more complex system. Then, the keys generated by a QKD protocol with universally-composable security may be safely used within any application, such as message encryption and communication. This form of security is usually assessed by the distance of a real cryptographic protocol to an ideal protocol, which is perfectly secure. We define an ideal QKD protocol as one which either aborts without generating a secret key or generates an ideal secret key, i.e., a uniformly-random bit string shared between Alice and Bob and independent of any information Eve might have collected.

According to this definition, a QKD protocol, described by the CPTP map composed of local operations and classical communication QKD_{real}, is said to be β-secure if

$$\|QKD_{real} - QKD_{ideal}\|_{\diamond} \leq \beta, \tag{2.17}$$

where QKD_{ideal} describes its equivalent ideal protocol. According to equation (2.11), we can express this condition in terms of the output of the protocols with

$$\sup_{\rho_{ABE}} \|QKD_{real}(\rho_{ABE}) - QKD_{ideal}(\rho_{ABE})\|_{tr} \leq \beta, \quad \text{for EB-QKD,} \tag{2.18}$$

$$\sup_{\mathcal{N}_{A\to BE}} \|QKD_{real}(\mathcal{N}_{A\to BE}) - QKD_{ideal}(\mathcal{N}_{A\to BE})\|_{tr} \leq \beta, \quad \text{for PM-QKD,} \tag{2.19}$$

where ρ_{ABE} ranges over all possible input quantum states shared between Alice and Bob and its extensions to an auxiliary system held be Eve, and $\mathcal{N}_{A\to BE}$ ranges over all quantum channels from Alice to Bob and Eve. Then, establishing security implies showing that the trace distance is small for all possible protocol inputs. Moreover, condition (2.19) for a PM-QKD protocol is equivalent to condition (2.18) for its idealized EB-QKD version [22].

As shown in [50], this condition may be split into two conditions corresponding to correctness and secrecy. The correctness condition relates to the probability of Alice and Bob sharing an identical key. It corresponds to an upper bound on the probability of the protocol not aborting, but outputting distinct final keys for Alice and Bob. A QKD protocol is said to be ϵ_{cor}-correct if, for every protocol input,

$$P\left(\boldsymbol{k}_A \neq \boldsymbol{k}_B \wedge F^{abort} = 0\right) \leq \epsilon_{cor} \in [0,1), \tag{2.20}$$

where F^{abort} is the flag which indicates the protocol aborted if it is set to 1. The secrecy condition relates to the probability of the final keys being independent of any information Eve obtained. It corresponds to an upper bound on the trace distance between the output of the real protocol when the final keys are identical and the protocol did not abort and the output of an ideal protocol which also did not abort. A QKD protocol is ϵ_{sec}-secret if, for every protocol input,

$$\left\| QKD_{real \mid \boldsymbol{k}_A = \boldsymbol{k}_B \,\wedge\, F^{abort}=0}(\,\bullet\,) - QKD_{ideal \mid F^{abort}=0}(\,\bullet\,) \right\|_{tr} \leq \epsilon_{sec} \in [0,1), \tag{2.21}$$

where $\bullet$ indicates the protocol input according to the category of QKD protocol. Note that both of these conditions consider only the cases when the protocol did not abort, since the output of the real and ideal protocols when they abort are identical. Then, a QKD protocol is β-secure with $\beta = \epsilon_{cor} + \epsilon_{sec}$ if it is ϵ_{cor}-correct and ϵ_{sec}-secret. This framework allows us to establish trade-offs between protocol parameters such as block length, secret-key length and tolerated error rate, and the security parameters ϵ_{cor}, ϵ_{sec} and β.

[50] also introduced generalizations of the von-Neumann entropy, called smooth min- and max-entropies. As with the von-Neumann entropy, these measures provide ways of quantifying the uncertainty that an observer has on the state of quantum system. The smooth min-entropy quantifies how much uniform randomness can be extracted from a quantum system and the smooth max-entropy corresponds to the minimum number of bits needed to encode the system's state in a way that it can be recovered without errors, being related to data compression. Based on the notion of universal composability and using the smooth min- and max-entropies, they proposed a general framework for analysing the security of QKD protocols with finite resources. In this framework, the conditions for universally-composable security can be satisfied by deriving a sufficiently-strong lower bound on the smooth min-entropy of Alice and Bob's corrected key conditioned on Eve's side information. They also established the first security proof in that regime, providing trade-offs between protocol and security parameters. This was the first rigorous framework applicable to CV-QKD protocols.

The trade-offs in [50] were significantly improved in works such as [22, 51, 52]. The latter works finally provided sufficiently-tight proofs, i.e., proofs which ensure information-theoretical security of QKD protocols with realistic parameters and implementations. In particular, [22, 51] used a generalization of the entropic uncertainty relations to smooth min- and max-entropies [53], which is now the most common approach in QKD security analysis [5]. [22] provided the best parameter trade-offs to date for the families of protocols presented in sections 2.2.1 and 2.2.2. According to their analysis, for a version of the above family of EB-QKD protocols where both parties already share an entangled state at the beginning, we can bound the conditions (2.20) and (2.21) by

$$\epsilon_{cor} = 2^{-t}, \tag{2.22}$$

$$\epsilon_{sec} = \inf_{\nu \in \left(0, \frac{1}{2} - \delta\right)} \epsilon_{pe}(\nu) + \epsilon_{pa}(\nu),$$

$$\text{with} \quad \epsilon_{pe}(\nu) = 2\exp\left(-\frac{(m-k)k^2\nu^2}{m(k+1)}\right) \quad \text{and} \quad \epsilon_{pa}(\nu) = \frac{1}{2}\sqrt{2^{-(m-k)\left(\log_2\left(\frac{1}{c}\right) - H(\delta+\nu)\right) + r + t + l}}, \tag{2.23}$$

respectively, where $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is the binary Shannon entropy and $\bar{c}$ is the average overlap of Alice's measurements, as described above in the common assumptions of these analyses. The parameter ν can be optimized numerically. We note note that ϵ_{cor} vanishes asymptotically as we increase the length t of the hash used for error correction, which should increase with the block length m. ϵ_{sec} also vanishes as we increase m, for specific choices of protocol parameters. In particular, the optimal choice of parameters allows us to achieve the asymptotically-optimal key rate of

$$\frac{l}{m} = \log_2\left(\frac{1}{\bar{c}}\right) - 2H(\delta). \tag{2.24}$$

This is the largest key rate for which the above expressions still vanish for large m. Since they also proved that the defined family of PM-QKD protocols can be reduced to this version of the family of EB-QKD protocols, these trade-offs also apply to the former, with two modifications: $\bar{c}$ is replaced by the complementarity measure of the prepared states $\bar{c}'$ and the key rate is given by $\frac{l}{M} = \frac{m}{M}\frac{l}{m}$. Therefore, if a such an EB-QKD protocol, parametrized by m, pe, ec, pa, is β-secure with average measurement overlap $\bar{c}$, then a protocol of the PM-QKD family parametrized by M, m, pe, ec, pa, with state-complementarity parameter $\bar{c}' = \bar{c}$, is also β-secure. The factor $\frac{m}{M}$ may be increased by optimizing the sifting step of these protocols with strategies which do not fix M in advance, as discussed in [54].

More recently, [55] proposed a new analysis technique based on entropy accumulation. This technique uses the fact that the smooth min-entropy of the total quantum system accumulates in each round of a QKD protocol and approximates it by the sum of the von-Neumann entropies of its individual subsystems. This technique has been mainly used to prove the tight finite-resource security of DI-QKD protocols [56], which are discussed in section 2.3.1. Nevertheless, these trade-offs are still not as strong as the ones achieved with entropic uncertainty relations [5], even though they have been recently improved in [57].

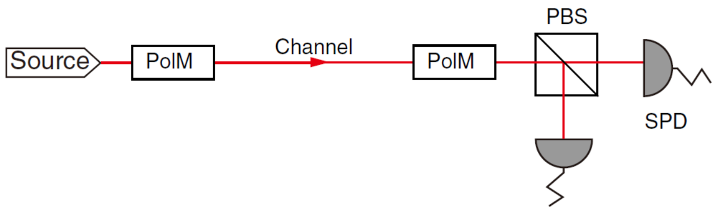
Various authors [5, 22, 58] believe there is room for improving the best current trade-offs through further collaboration between theory and experiment. Specifically, there are many protocols for which techniques such as entropic uncertainty relations or entropy accumulation do not currently yield optimal key rates. Some examples are the six-state protocol and DI-QKD protocols [5]. There is also room for refinement of the mathematical models behind security proofs to better describe the quantum hardware used in implementations.

2.2.4 Implementations of QKD

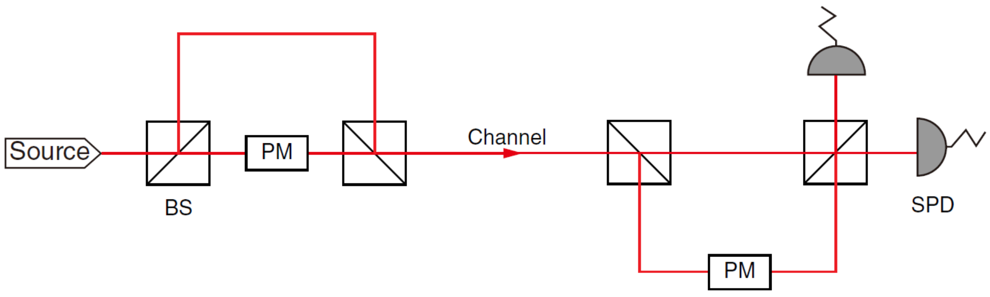
As mentioned in section 2.2.3, the security of a QKD protocol is often related to its implementation. The crucial components of a QKD implementation are the source, the encoder, the quantum channel, the decoder and the measurement device. Here, we describe the most common physical implementations of DV-QKD, along with the main algorithms for classical postprocessing, focusing on error correction and privacy amplification. We also delve into classical and quantum repeaters as a way of extending the transmission distance of QKD implementations.

Physical implementations

Photons are the most-commonly-used particles for encoding qubits in QKD, since they are fairly robust to decoherence due to noise and travel at a high speed [58]. Qubits are most often encoded in the polarization states of photons or in the relative phase between superposition states of a photon. The former scenario is usually known as polarization encoding and the latter goes by several names such as phase, path, interferometric or time-bin encoding [5, 58]. Examples of implementation schemes for both encodings are shown in figures 2.1a and 2.1b. These implementation schemes can be used for multiple variants of the BB84 protocol, for instance. Some implementations also encode qubits in the relative phase between sidebands of a central monochromatic pulse, which is known as frequency encoding [59]. However, these implementations are much less common.



(a) A polarization-encoding QKD implementation.



(b) A phase-encoding QKD implementation.

Figure 2.1: Examples of basic schemes for polarization- and phase-encoding PM-QKD implementations. PolM: polarization modulator; PM: phase modulator; PBS: polarization beam splitter; BS: beam splitter; SPD: single-photon detector.

Source The source used in DV-PM-QKD protocols is usually modeled as a single-photon source [58]. However, the practical realization of such a source is still a difficult experimental challenge. Therefore, sources which emit weak coherent states with varying number of photons are usually used in implementations. For DV-EB-QKD, it is common to use sources which use parametric down-conversion to produce entangled photon pairs [6]. In these sources, each high-frequency photon generated is converted into an entangled pair of low-frequency photons.

Encoder In PM-QKD, after the quantum systems are generated by the source, classical bits are encoded into their quantum states by a modulator. In a typical polarization-encoding implementation, a polarization modulator (PolM in figure 2.1a) is used set the desired polarization of each photon. Most state-of-the-art implementations use electro-optic modulators which change a photon's polarization when a specific voltage is applied [60]. Instead of a polarization modulator, some polarization-encoding implementations use multiple sources which already emit photons in the desired polarizations [6]. In each round, only one of these sources is triggered, according to the chosen basis and classical bit, and the source outputs are then combined into one channel with beam splitters (BS). In a usual phase-encoding implementation, the photons go through a beam splitter and enter a superposition of states going through two different paths [60]. One of the paths has a phase modulator (PM) which usually uses electro-optics to change the optical path length of that path with great precision. This introduces a phase between the two states. The two paths then recombine in another beam splitter so that the photons enter a superposition of states with the desired relative phase between them, which encodes the selected bit. This encoder setup corresponds to an unbalanced Mach-Zehnder interferometer [61].

Quantum channel In a rigorous security analysis, Eve is assumed to fully control the quantum channel. Therefore, a protocol's security does not depend on the physical implementation of the quantum channel. However, the channel's quality, i.e, its loss, does affect the protocol efficiency. In a typical optical PM-QKD experiment, the key rate is expected to be $\frac{l}{M} = \frac{\eta l}{2m}$, with η being the overall transmittance of the quantum channel [22]. The most-commonly-used quantum channels are standard to ultra-low-loss optical fibers [7] and free space [62]. Since polarization states are not very robust to decoherence in optical fibers due to their high polarization dispersion, most implementations with fibers use phase encoding [58]. On the other hand, free-space implementations commonly use polarization states as qubits, which are particularly robust to decoherence in that medium.

Decoder The role of the decoder setup is to apply a measurement on the received quantum systems according to a chosen basis. In polarization-encoding implementations, a polarization modulator is commonly used to project the incoming photons into a state with a randomly-chosen polarization basis [60]. In case the encoding basis is chosen, the polarization modulator does not alter the photon's state. The photons then go through a polarization beam splitter (PBS) which directs them through two paths according to their polarization. These paths are connected to single-photon detectors (SPD) whose detection corresponds to a particular classical bit. In some implementations, instead of a polarization modulator, the basis measurement can be applied through a sequence of beam splitters and waveplates which rotate the photons' polarization by a predefined amount [6]. In usual phase-encoding implementations, the decoder setup also corresponds to an unbalanced Mach–Zehnder interferometer [60]. A phase modulator in one of the paths is configured so that the interferometer applies a measurement in the randomly-chosen phase basis by adding an additional relative phase to the superposition state of a photon. The second beam splitter of the interferometer is connected to two single-photon detectors which detect each photon according to a probability dependent on the relative phase of its state.

Detectors Ideal single-photon detectors are also an experimental challenge [58]. The commonly-used detectors are avalanche photodiodes and superconducting single-photon detectors [60]. In avalanche photodiodes, an incident photon triggers an electron-avalanche effect which produces a detectable voltage pulse. In superconducting single-photon detectors, an incident photon hits a superconductive wire, making that section of the wire non-superconductive and producing a voltage spike. These detectors usually act as threshold detectors, only being able to distinguish the vacuum state from single- and multi-photon states. However, there are also photon detectors capable of measuring the number of photons per state [63]. As mentioned before, all these detectors are subject to both photon loss and dark counts.

We note that several other implementation setups have been proposed and realized, such as phase-encoding implementations based on Michelson interferometers with Faraday mirrors [64], implementations which use phase encoding and polarization decoding [65] and plug-&-play phase-encoding implementations [66], which automatically compensate for both polarization and phase drift in optical fibers.

Such implementations are used to perform all quantum processing of most QKD protocols. The remaining processing, particularly error correction and privacy amplification, is currently classical.

Error correction

The error-correction step of a QKD protocol, also known as information reconciliation, is used to correct differences between the raw keys of Alice and Bob so that they share identical corrected keys. Classical error correction is performed over the authenticated public channel. Thus, it is essential to leak the minimum amount of information about the raw keys. This is done by sharing only redundant information, such as bit-parity data, i.e, the parity of the number of ones in chosen key blocks. In practice, the most-commonly-used algorithms for error correction are the cascade protocol [67] and forward-error-correction (FEC) codes.

Cascade protocol The cascade protocol is a highly-interactive protocol, i.e., it involves two-way communication between the parties in multiple steps. It uses a primitive, first described in [68], which consists in an interactive binary search which detects a single-bit error in two strings with an odd number of errors by exchanging the parity of specific substrings. For strings of length n, this primitive detects a single-bit error by exchanging at most $\lceil \log_2(n) \rceil$ bits over the public channel. In each iteration of the cascade protocol, the raw keys are shuffled, equally divided into blocks of an agreed length and the parity of each block is computed. Then, Alice sends her parities to Bob, which compares them with his. They run the interactive binary search in each block pair where their parities differ and Bob corrects the errors found in each of his blocks. For each corrected bit, the blocks from previous rounds which contained it will then have their parity switched. Thus, the same binary search must also be performed on all those previous blocks. This procedure is repeated recursively for blocks where the parity between Alice and Bob differs. Therefore, each bit correction leads to a cascade of bit corrections, which is also known as traceback step or cascade effect and gives the name to this protocol. The protocol usually runs for a predefined number of iterations and Alice and Bob end up with identical corrected keys with high probability.

The block length for each iteration is usually also determined before executing the cascade protocol and tends to be related to the estimated error rate of the quantum channel Q. For instance, the original version of the cascade protocol empirically proposed four iterations and block lengths $n_1 = \left\lceil \frac{0.73}{Q} \right\rceil$ for the first iteration and $n_i = 2n_{i-1}$ for the following [67]. This choice makes the probability of a block containing an error decrease exponentially with the increase in the number of iterations.

Several optimizations of the cascade protocol have been proposed in the last two decades, either focusing on improving the choice of parameters [69, 70] or on modifying the protocol steps [71–73].

Nevertheless, the modifications analysed and proposed in [71–73] offer trade-offs between the number of public-channel uses and the computational resources of both parties. Other interactive protocols have also been proposed as an alternative to the original cascade protocol, such as the Winnow protocol [74], which reduces the required number of public-channel uses and simultaneously performs privacy amplification. Various FEC codes were also proposed as an alternative to the cascade protocol.

FEC only requires Alice to send one error syndrome to Bob, greatly reducing the public-channel use in comparison to interactive protocols [58]. Nevertheless, this syndrome contains a lot more information about the key than the bits exchanged in one round of interactive protocols. FEC was used in the families of QKD protocols defined in sections 2.2.1 and 2.2.2. The most-commonly-used FEC codes in QKD are low-density parity-check (LDPC) codes [75].

LDPC codes LDPC codes are linear codes which can be defined by a sparse parity-check matrix. In the QKD scenario, a parity-check matrix is a binary matrix which describes the parity of specific substrings of the keys allowed by the LDPC code. A LDPC code described by a $(n-d) \times n$ parity-check matrix is known as a $[n, d]$ LDPC code. The set of $n-d$ linear relations described by the parity-check matrix may also be represented by a bipartite graph, also known as Tanner graph [76], where a n-node set represents the columns of the parity-check matrix, a $(n-d)$-node set represents its rows and edges between two nodes represent its non-zero entries. For instance, the parity relations

$$\begin{cases} k_0 + k_1 + k_3 = 0 \mod 2 \\ k_1 + k_2 = 1 \mod 2 \\ k_0 + k_4 = 0 \mod 2 \end{cases} \iff \begin{cases} k_0 + k_1 + k_3 = 0 \mod 2 \\ k_1 + k_2 + 1 = 0 \mod 2 \\ k_0 + k_4 = 0 \mod 2 \end{cases} \tag{2.25}$$

between the bits k_i of the allowed keys can be used to build the parity-check matrix

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.26}$$

and its corresponding Tanner graph, shown in figure 2.2.

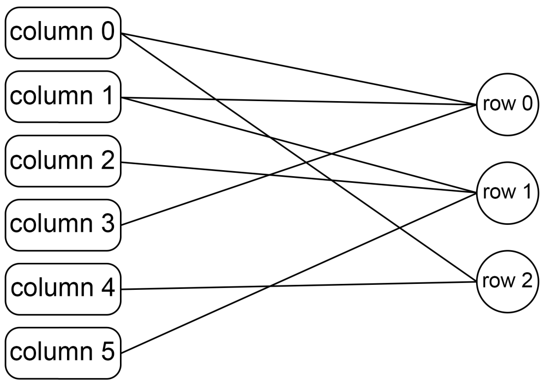


Figure 2.2: Tanner graph corresponding to the parity-check matrix (2.26).

A key allowed by a LDPC code with parity-check matrix $\boldsymbol{H}$ can be used to build a vector $\boldsymbol{k}$ which satisfies

$$\boldsymbol{H} \cdot \boldsymbol{k} = 0 \mod 2, \tag{2.27}$$

where the modulo refers to the value of each entry of the resulting vector. In coding theory, such vectors are known as the codewords of the code [77]. A key which is not allowed by the code relations will generate a vector $\boldsymbol{k}'$, which will differ from one of the codewords $\boldsymbol{k}$ by a few entries such that we can write $\boldsymbol{k}' = \boldsymbol{k} + \boldsymbol{e} \mod 2$. Then, a syndrome of vector $\boldsymbol{k}'$ is given by

$$\boldsymbol{H} \cdot \boldsymbol{k}' = \boldsymbol{H} \cdot (\boldsymbol{k} + \boldsymbol{e}) = \boldsymbol{H} \cdot \boldsymbol{e} \mod 2. \tag{2.28}$$

Therefore, exposing the syndrome will not reveal which codeword, and consequently which raw key, Alice has.

For QKD, the choice of parity-check matrix is crucial to maximize the efficiency of error correction and minimize the information leakage [77]. The length of the codewords, the number of non-zero entries in the parity-check matrix and the parity relations should be chosen according to the average error rates of the quantum channel and measuring devices. One of the most successful algorithms for constructing sparse parity-check matrices is known as progressive edge growth [78] and was improved in [79].

In a QKD protocol using LDPC codes, Alice and Bob first estimate the error rate of the quantum channel and generate a sparse parity-check matrix $\boldsymbol{H}$. Then, Alice generates a vector $\boldsymbol{k}'$ from her raw key according to the rules defined by the LDPC code and sends the syndrome $\boldsymbol{H} \cdot \boldsymbol{k}' \mod 2$ to Bob over the public channel. With his raw key and Alice's syndrome, Bob uses a suitable decoding algorithm to obtain a corrected key identical to Alice's. This can be done using iterative belief-propagation algorithms [80], also known as sum-product algorithms, which can decode Alice's raw key with high probability in time proportional to the number of non-zero entries in the parity-check matrix. Bob runs such algorithm until his key is identical to Alice's with high probability or until he reaches a predefined number of iterations, after which he aborts the protocol.

Several optimizations of LDPC codes adapted to the QKD scenario have been proposed, from which we highlight a few. In [81], an optimization of LDPC codes to a specific type of quantum channel, known as binary symmetric channel, was proposed. [77] proposed a protocol which adapts to changes in the error rate of the quantum channel in each round by using puncturing techniques, which correspond to removing bits from the codewords, and shortening techniques, which correspond to predefining some of the bits of every codeword, thus removing them from the encoding process. This protocol also includes a final interactive step which is meant to increase its success rate. [82] adapted the techniques of the protocol in [77] to short codeword lengths, making it feasible for practical hardware implementations. It also turned the protocol into a blind protocol, i.e., a protocol which does not need an initial estimation of the quantum-channel error rate, thereby saving additional bits of the raw key. [83] improved the efficiency of the previous blind protocol based on the symmetry of its operations. Finally, [84] extended the protocol in [83] to use multiple parity check matrices, and therefore multiple syndromes, which provides better estimations of the quantum-channel error rate, reduces the number of iterations needed for decoding and has a higher success rate. Alternative FEC codes have also been proposed, such as turbo codes [85] and polar codes [86].

In general, the cascade protocol leaks less bits than LDPC codes for low quantum-channel error rates, while the opposite is true for high error rates [87]. The main limitation of LDPC codes is the high computational cost of the iterative decoding. Thus, it is usually performed with graphics processing units [58]. Nevertheless, the low latency ensured by the reduced use of the public channel makes FEC codes suitable candidates for commercial QKD implementations.

We note that, after using one of the above algorithms for error correction, Alice and Bob can verify the success of the procedure with a hash function from a universal$_2$ family, as described in section 2.2.1.

Privacy amplification

The error-correction process necessarily leaks information about the corrected key to Eve. Moreover, she may also have obtained additional information by eavesdropping on the quantum channel. Therefore, a procedure is required to extract only the secret bits from the corrected key. This distillation of a secret key from a partially-exposed key is called privacy amplification and was first introduced in [88]. It directly relates to the secrecy condition (2.21) of universally-composable security for QKD protocols. As with classical error correction, classical privacy amplification is performed over the authenticated public channel.

Using universal$_2$ families of hash functions for privacy amplification, as done in the QKD protocols of sections 2.2.1 and 2.2.2, was first proposed in [88], where it was considered the case in which Eve obtained deterministic classical information about the corrected key. [89] extended the privacy-amplification framework to the case where Eve obtained probabilistic classical side information about the corrected key. Later, [50] generalized this framework to the case where Eve has a quantum memory, i.e., she is allowed to keep her information in the form of a quantum state, which she can entangle with Alice and Bob's qubits, and she can measure it at the end of the QKD protocol, after knowing Alice and Bob's choice of hash function. Finally, [90] extended this framework to the use of δ-almost universal$_2$ families of hash functions, which allow the use of shorter random seeds and can lead to a speedup in practical implementations at the expense of a very small reduction in the size of the final secret key.

In general, linear privacy amplification, such as matrix hashing, is commonly used in QKD due to its low computational complexity [58]. Let $h_{\boldsymbol{M}}(\boldsymbol{x}) = \boldsymbol{M} \cdot \boldsymbol{x} \bmod 2$ be the logical multiplication of a matrix $\boldsymbol{M} \in \{0,1\}^l \times \{0,1\}^n$ and a vector $\boldsymbol{x} \in \{0,1\}^n$. Then, the set $\{h_{\boldsymbol{M}}\}_{\boldsymbol{M}}$ is a universal$_2$ family of hash functions [23] and could be used for privacy amplification. However, since these functions are parametrized by $l \times n$ values, a random seed for such a function which extracts l bits from n bits would have $l \times n$ bits. We can obtain other much smaller universal$_2$ families by imposing specific restrictions on the entries of these matrices. In QKD, the most-frequently-used universal$_2$ hash functions are based on binary Toeplitz matrices [91].

A Toeplitz matrix has entries which are constant along each of its diagonals. Thus, a $l \times n$ Toeplitz matrix has the form

$$\boldsymbol{T} = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & & \\ a_2 & a_1 & a_0 & & \vdots \\ \vdots & & & \ddots & \\ a_{l-1} & & \cdots & & a_{l-n} \end{bmatrix}. \tag{2.29}$$

It can also be written concisely as $T_{i,j} = a_{i-j}$, where $T_{i,j}$ is the entry (i, j) of $\boldsymbol{T}$. Such a matrix can be parametrized by only $l + n - 1$ values, leading to similar hashing with much shorter seeds.

More recently, [92] showed that the concatenation of a Toeplitz matrix with an identity matrix, commonly referred to as modified Toeplitz matrix, also forms a universal$_2$ family of hash functions. With this family, in order to extract l from n bits, we would generate a $l \times (n - l)$ binary Toeplitz matrix $\boldsymbol{T}$ and concatenate it with a $l \times l$ identity matrix $\mathbb{I}$ as $(\boldsymbol{T}, \mathbb{I})$, for instance. Thus, we only need to generate a random seed with $n - 1$ bits to choose a modified Toeplitz matrix.

In a QKD protocol using Toeplitz matrices, Alice and Bob first estimate an upper bound on the side information that Eve may have on the corrected key. Let the corrected key have n bits and that upper bound be $n - l$ bits. Then, they randomly choose a $l \times n$ binary (modified) Toeplitz matrix $\boldsymbol{T}$ and share it over the authenticated public channel. Then, by applying the chosen matrix on their corrected keys, $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$, they will get identical secret keys, $\boldsymbol{k}_A = \boldsymbol{T} \cdot \boldsymbol{x}_A \mod 2$ and $\boldsymbol{k}_B = \boldsymbol{T} \cdot \boldsymbol{x}_B \mod 2$, with high probability.

Repeaters

As mentioned before, simple QKD implementations are still fairly limited in transmission distance due to quantum-channel losses. These losses lead to decoherence of the transmitted quantum systems, which become noisy and lose their information. In order to overcome this, repeaters may be used along the communication channels. In classical communications, repeaters work by amplifying the classical signal. This requires copying the signal. However, quantum states cannot be determined, and thus cannot be copied. Such an attempt requires measuring the state and therefore collapses it, destroying its information. This is known as the no-cloning theorem [93] and is also the reason Eve cannot eavesdrop on the quantum channel without being detected. Thus, QKD repeaters must work differently.

Classical repeater The simplest type of QKD repeater is a trusted classical repeater [8], which is composed of a receiver and a sender device. Instead of transmitting qubits over long distances, these repeaters work by key composition. In a QKD network with one trusted classical repeater between Alice and Bob, the following protocol is performed:

1. Alice and the repeater perform the QKD protocol to generate the shared key $\boldsymbol{k}_{AR}$;
2. Bob and the repeater perform the QKD protocol to generate the shared key $\boldsymbol{k}_{RB}$;
3. The repeater encrypts $\boldsymbol{k}_{AR}$ with $\boldsymbol{k}_{RB}$, for instance as $\boldsymbol{m} = \boldsymbol{k}_{AR} + \boldsymbol{k}_{RB} \mod 2$, and sends it to Bob;

4. Bob decrypts the received key by performing $\boldsymbol{k}_{AR} = \boldsymbol{m} + \boldsymbol{k}_{RB} \mod 2$ and now shares the key $\boldsymbol{k}_{AR}$ with Alice.

The key is secure from Eve as long as the repeater is trusted, since it also knows the key $\boldsymbol{k}_{AR}$. This trust can be relaxed with the use of DI-QKD protocols [94], which are discussed in section 2.3.1. The above protocol is easily generalizable to the multiple-repeater case. There have been multiple real-world implementations of QKD networks with trusted classical repeaters, such as [95–97].

Quantum repeater Another form of QKD repeater is an untrusted quantum repeater [9]. These repeaters can be used to transmit qubits over long distances and are mostly designed for EB-QKD. Quantum repeaters can be seen as small quantum computers which usually rely on entanglement distillation, also known as entanglement purification, to correct for the decoherence of the quantum channel. This consists in transforming a large number of qubit pairs, which became less entangled due to noise, into a smaller number of qubit pairs with higher entanglement through local operations and classical communication. It is done by entangling Alice's qubits with qubits on the repeater and entangling another set of qubits on the repeater with Bob's qubits. Then, local operations are performed on all qubits of the repeater, and the results communicated to Bob, such that Alice's qubits become entangled with Bob's qubits. This is also referred to as entanglement swapping [98]. Networks with multiple quantum repeaters may perform entanglement swapping in a nested fashion [9]. Since these repeaters establish entanglement directly between Alice and Bob, and this entanglement can be tested for with specific measurements, the security of networks with such repeaters does not rely on the repeaters.

Quantum repeaters can also rely on quantum-error-correction codes [99]. However, such repeaters would require a very large amount of entangled qubits and are currently impractical.

The recent proof-of-concept experiment performed by [100] is the first physical realization of a quantum repeater. However, there are no networks with quantum repeaters implemented outside a laboratory yet [5].

2.3 Side channels

Even though QKD is information-theoretically secure, its implementations may not be. As mentioned in section 1.1, this occurs due to unintended side-channel leakages in the physical devices used in the protocols being correlated with the internal state of such devices, which is itself related to the generated key. By interacting with and exploiting these physical side channels rather than the abstract protocol, Eve can obtain information about the generated key without being detected. Hence, all cryptographic implementations must take the possible side-channel attacks into account to guarantee security.

In classical cryptographic implementations, commonly exploited side channels are the power consumption, or power trace, of the used devices [10], the electromagnetic radiation emanated from them [11], the sound produced by them during the protocol [12], their cache state during the protocol [101] and the time taken to perform specific steps of the protocol [102], for instance. In QKD, the most exploited side channels are related to the quantum processing of the protocols, namely the state preparation, distribution and measurement [5]. Nevertheless, the devices used in most steps of a QKD protocol, from state preparation and measurement to error correction and privacy amplification, may also be vulnerable to the side channels common in classical implementations, even though these are not taken into account often. As mentioned in section 1.1, we will refer to the latter as classical side channels and to the ones related to the quantum processing of QKD protocols as quantum side channels.

Since side channels are directly related to the devices used in implementations, side-channel attacks are generally implementation specific. Side-channel attacks can be divided into two phases [103]. The first is an optional preparation phase in which Eve profiles and characterizes the leakages with a possibly-simulated device similar to the one on which she intends to perform the attack. The second phase is the exploitation phase in which she performs the attack on the real target device with the intent of extracting information about the secret key.

In general, when the key length is large, fully characterizing the leakages usually requires processing very large amounts of data. Since this is typically unfeasible, a sample of those leakages is commonly used to design a side-channel attack [103]. The best way to characterize a side channel from a sample of leakages is still an open problem.

An attack based on a probability distribution built from a sample of leakages is known as a template attack [104] and is assumed to be the strongest type of side-channel attack. Therefore, these are commonly used to evaluate the security limits of devices with leakages.

2.3.1 Side-channel attacks in QKD

Here, we review some of the main side-channel attacks demonstrated in QKD implementations and proposed countermeasures. We start with quantum-side-channels attacks due to chronological order.

Quantum-side-channel attacks and countermeasures

There have been several quantum attacks proposed in the literature, targeting both source and detection components. Let us start with a description of some of the main source-targeting quantum attacks and their countermeasures.

Photon-number-splitting attack The first proposed attack to target source components was the PNS attack [41]. In a PNS attack, Eve takes advantage of the coherence of multi-photon sources. She first measures each photon pulse sent through the quantum channel to know its photon number. If the pulse contains a single photon, she blocks it. Otherwise, she keeps a few photons of each pulse stored in a quantum memory. Since Bob only measures undisturbed photons, she remains undetected. After all the classical communication is performed, Eve can measure her stolen photons according to the information disclosed through the public channel to obtain information about the secret key. For instance, in the BB84 protocol, Alice reveals all measurement bases, so Eve can use them to obtain the raw key and perform the same postprocessing as Alice and Bob to acquire their secret key.

As mentioned before, the SARG04 protocol was proposed as PNS-resistant protocol. In it, instead of directly revealing her bases, Alice reveals pairs of non-orthogonal states in which her bits might have been encoded. This way, Eve does not know which bases to use for her photons. Another countermeasure to the PNS attack are protocols which use decoy states to detect it [105]. In these protocols, Alice prepares pulses with varying intensity on purpose. The pulses with higher intensity contain more photons and are decoy states. After Bob measures the pulses, Alice reveals the intensities she used for each pulse and Bob can compute the detection rate for each intensity. If Eve performed a PNS attack, she blocked single-photon pulses, therefore changing the detection rates for signal and decoy states differently. Therefore, she is forced to let most single-photon pulses pass to avoid being detected and cannot known most key bits. Currently, most implemented QKD protocols use decoy states [58].

Phase-remapping attack In the phase-remapping attack on phase-encoding implementations [106], Eve may use an optical delay line on the quantum channel to change the arrival time of pulses on the phase modulators such that she can bias the encoded phase in each pulse. This allows her to intercept and resend pulses while introducing an error rate small enough not to be detected. This way, Eve can obtain most, if not all, of the secret key. Therefore, in implementations vulnerable to this attack, Alice and Bob must decrease their allowed error rate to a value low enough to detect this attack. This leads to a significant decrease in the key rate of the protocol.

Laser-seeding attack The laser-seeding attack [107] works on semiconductor-laser-diode sources, which are the most used sources in both research and commercial QKD systems [58]. In this attack, Eve injects photons from an external source into the setup source through the quantum channel. If these external photons have enough energy, they trigger pulses from the setup source and are thus known as seed photons. By controlling the phase of the injected photons, Eve can control the phase of the triggered pulses. This violates the common assumption that the phase of generated pulses is uniformly random and Eve can use this knowledge to obtain the secret key without being detected. A countermeasure is to actively randomize the phase of the pulses after generating them.

In QKD implementations, detection components are generally more vulnerable than source devices [58]. Most detector-targeting attacks are faked-state attacks [108]. These are a type of intercept-and-resend attack which exploits detector imperfections. In them, instead of trying to reconstruct the original state, Eve resends carefully-prepared states which allow her to infer the detection results. We now describe some of the main quantum attacks targeting detection components.

Faked-state attack (1) This faked-state attack [109] exploits the detection-efficiency mismatch of two detectors. A slight time-dependent mismatch is common in real implementations due to the finite manufacturing precision in detection components. Such mismatch can also be induced by Eve during calibration of the implementation [110]. In this attack, Eve intercepts each pulse from the source and measures it in a random basis, obtaining a bit. Then, she resends a state encoding the opposite bit prepared in the basis opposite to the one she chose. By carefully timing this process, she can send the faked state when the detector which measures that opposite bit has a lower detection efficiency than the other detector. This way, by increasing the rate of inconclusive measurements, Eve can lower the error rate and obtain the full secret key without being detected. Thus, it is crucial to fully characterize the detectors and either actively check the timing of incoming pulses or randomly shift the detection time window of the detectors, for instance.

Time-shift attack Based on the faked-state attack (1), the time-shift attack was proposed [111]. This attack is a simpler version of the faked-state attack (1) which also exploits the detection-efficiency mismatch of two detectors. In it, Eve uses an optical delay line on the quantum channel to change the arrival time of pulses such that they arrive when one of the detectors has a much lower detection efficiency than the other. This way, if Bob reports a conclusive measurement, Eve will know which bit he measured with high probability. Moreover, Eve can shift half of the pulses forward and half backward in time to preserve the distribution of ones and zeros such that she is not detected. The same countermeasures for the faked-state attack (1) can be applied for this attack.

Polarization-shift attack The polarization-shift attack [112] also exploits the detection-efficiency mismatch, particularly of superconducting single-photon detectors in phase-encoding implementations. These detectors have a polarization-dependent detection efficiency and a slight mismatch is common due to the finite precision of real implementations. Thus, Eve can find two polarizations such that one detector has a much higher detection efficiency than the other, and vice-versa. This way, she can use a polarization modulator in the quantum channel to rotate the polarization of pulses to one of those two polarizations. The rest of the attack proceeds similarly to the time-shift attack. A countermeasure is to use a polarization filter in the decoder setup to filter out problematic polarization components.

Detector-blinding attack The detector-blinding attack [113] is another faked-state attack. In it, Eve sends intense continuous light to the avalanche-photodiode detectors. This forces the detectors to work in linear mode, where they do not detect single photons. In linear mode, these detectors produce currents proportional to the intensity of the input pulses. This operation is called detector blinding. After blinding the detectors, Eve intercepts each pulse from the source and measures it in a random basis, obtaining a bit. Then, she resends a state encoding the opposite bit prepared in the basis opposite to the one she chose, as in the faked-state attack (1). By carefully calibrating the pulse intensity, setting it in the order of millions of photons, the following scenarios occur if Eve uses the same basis as Alice and detects the correct bit:

- If Bob also chooses the same basis, then one of his detectors will register a photon and Alice, Bob and Eve will share the same bit;
- If Bob chooses the opposite basis, the optical power of the pulse will be equally divided between the two detectors and the voltage spike in both detectors will be below the detection threshold. Thus, he will register a photon loss.

On the other hand, if Eve uses the basis opposite to Alice's, then:

- If Bob chooses the same basis as Eve, he will have chosen the basis opposite to Alice's;
- If Bob chooses the basis opposite to Eve's, he registers a photon loss.

After sifting the bits and discarding inconclusive and opposite-basis measurements, Alice, Bob and Eve will share similar raw keys and, after postprocessing, Eve will have obtained the secret key without being detected. This attack is also applicable to superconducting single-photon detectors [114]. A possible countermeasure is to use a power meter to monitor the power of the pulses at the entrance of the decoder setup.

After-gate attack This attack [115] is another faked-state attack which proceeds very similarly to the detector-blinding attack and works on gated avalanche-photodiode detectors. However, instead of exploiting the linear mode of the detectors with intense continuous light, Eve does it by carefully timing the sending of the intensity-calibrated pulses. During the detection time window, the gate voltage of these detectors rises, plateaus and falls. In this attack, Eve sends the pulses after the gate voltage of the detectors falls and they enter linear mode but still register photons. The pulses must have a similar intensity to the ones used in the detector-blinding attack and similar countermeasures may be applied.

Superlinear-detector attack This attack [116] is also similar to the last two. In it, Eve sends the pulses during the rising or falling of the gate voltage, where the detectors are in superlinear mode. In this mode, Eve can achieve the same results with pulses of much lower intensity, on the order of 10 to 100 photons. Thus, monitoring the power of these pulses is much harder than in the previous two attacks. However, implementing this attack is also much harder due to the time precision required. Superconducting single-photon detectors also have a superlinear mode and are thus vulnerable to this attack [116]. Randomly shifting the detection time window of the detectors is a possible countermeasure.

Laser-damage attack Eve can go beyond blinding the detectors to damaging them with high-intensity continuous light [117]. This can permanently change the detectors' properties, leading to multiple possible attacks. For instance, Eve may be able to remotely control the detection efficiency and dark-count rate through damage, which violates the assumptions of most security proofs. Moreover, by intentionally lowering the dark-count rate, which leads to a lower error rate, Eve can more easily use faked-state attacks, for instance. Eve can also damage only one of the detectors to generate a detection-efficiency mismatch, which can be exploited with the faked-state attack (1) or the time-shift attack, for instance. Finally, Eve can use such intense light that it melts some components, creating an open circuit. This can be used to destroy power monitors before employing a detector-blinding attack. A possible countermeasure is to frequently characterize the implementation components during operation to ensure the validity of their initial characterization.

Detector-dead-time attack Between detection events, detectors need a finite amount of time to recover, in which they cannot register any photons. This is referred to as dead time. Eve can exploit the dead time of the detectors by timing pulses such that they arrive during the dead time after the previous detection [118]. This way, if both measurements were conclusive, they must be perfectly anticorrelated, corresponding to opposite bit values. Eve can do this for long pulse sequences, correlating them such that she only needs to guess the first bit of each sequence. This way, she is able to obtain a significant amount of information about the key without being detected. A possible countermeasure is to disable all detectors during the dead time of any of them. However, this is technically challenging at high transmission rates. Another option is to monitor the dead time of the detectors and discard all bits obtained during the dead time of any detector. In high-speed QKD, this can lead to a significant decrease in the key rate.

Detector-backflash attack After a detection event, avalanche-photodiode detectors have a probability of emitting backflash fluorescence light [119]. Eve can measure these backflash pulses in the quantum channel to gain information about which detector registered a photon. Possible countermeasures are using light filters to prevent the backpropagation of light with the typical properties of backflash light or using other types of detectors [5].

Wavelength-dependent attack This faked-state attack [120] explores the wavelength dependence of beam splitters used in the decoder setup. A calibrated beam splitter shows equal reflectivity and transmittivity for photons with the chosen wavelength. However, these properties usually depend on the wavelength of the photons, becoming highly unbalanced for certain wavelengths. Thus, Eve can send photons of specific wavelengths such that she controls the path they take on the decoder with high probability. This allows her to infer the raw key without significantly increasing the error rate. A countermeasure is to use a wavelength filter on the entrance of the decoder setup. Moreover, instead of passively relying on beam splitters, actively choosing the decoding measurement basis with a trusted quantum random-number generator is also an option.

A well-known attack which can target either source or detection components is the trojan-horse attack [121].

Trojan-horse attack The trojan-horse attack was initially known as large-pulse attack [122]. In it, Eve sends intense pulses to either the encoder or decoder setup. Then, she measures the low-intensity pulses that are naturally backscattered to know the basis choice of either setup or even which detector has registered a photon. Similarly to some of the previous attacks, a countermeasure is to monitor the power of the pulses entering the encoder and decoder setups. Another countermeasure is to use carefully-timed optical isolators to prevent pulses from entering the encoder setup and backscattered pulses from exiting the decoder setup. These isolators are usually wavelength dependent, so wavelength filters should also be used.

Several other attacks and corresponding security patches have been proposed and analysed [5]. We note that most proposed countermeasures can only reduce the leaked information, not eliminate it completely. Thus, the upper bound on Eve's information used in privacy amplification should take into account the possible remaining side-channel leakages. Moreover, security patches may introduce new side channels in the implementation.

Besides specific security patches, other approaches have been proposed based on modifying the QKD protocols themselves and refining the security proofs to remove as many side channels as possible from implementations. As mentioned in section 1.1, a known class of protocols resulting from these efforts is MDI-QKD [13]. MDI-QKD protocols are provably secure even with untrusted measurement devices. Thus, they remove all quantum side channels from the detection components, leaving only the source components possibly exposed. In MDI-QKD protocols, both Alice and Bob generate specific states which they send to an untrusted relay. Then, this relay measures both states simultaneously and shares the results with Alice and Bob through the public channel. These measurements are such that their results do not reveal Alice and Bob's shared bits. Moreover, Alice and Bob may verify the honesty of the relay by simply comparing a subset of the measurement results. MDI-QKD can be implemented with quantum channels as lossy and single-photon detectors as efficient as the ones used for standard QKD protocols, such as the decoy-state BB84 protocol [58]. In fact, MDI-QKD has already been successfully implemented in the real world [123]. While it yields considerably smaller key rates than standard QKD protocols, they are sufficiently large for use in real-world communications [58].

Another class of QKD protocols which removes side-channels from implementations is the already-mentioned DI-QKD [14]. These protocols are provably secure without the need to fully characterize the various implementation components. Thus, they arguably remove all quantum side channels from all implementation components. In DI-QKD protocols, self-testing of the devices is employed to guarantee their honesty. These tests are designed such that they can only be passed if the devices execute the protocol securely. In the most-well-known DI-QKD protocols, Alice and Bob assess security by testing an appropriate Bell's inequality, such as the CHSH inequality, for their entangled qubits. If this inequality is violated, quantum correlations are still present and indicate no tampering by Eve. DI-QKD is still highly impractical since it requires a nearly-lossless quantum channel and almost-perfect single-photon detectors to guarantee security [58]. Moreover, DI-QKD protocols yield much smaller key rates than MDI-QKD protocols [5]. The first proof-of-concept implementation of a DI-QKD protocol was realized over 2 meters of optical fiber only months ago [124].

Classical-side-channel attacks and countermeasures

There has been far less focus on preventing classical-side-channel attacks on QKD implementations. Here, we describe all proposals of such attacks and countermeasures so far, according to the targeted protocol step. Most of them are very recent.

Electromagnetic-radiation attack on state preparation There are two main categories of electromagnetic-radiation attacks, simple electromagnetic analysis (SEMA) and differential electromagnetic analysis (DEMA) [11]. SEMA involves interpreting the electromagnetic traces visually to determine which patterns correspond to which bits. DEMA involves statistically analysing multiple traces for the same purpose and is more robust against noise. In [18], they used SEMA to analyse the electromagnetic radiation emanated from the encoder of the plug-&-play phase-encoding implementation of the BB84 protocol demonstrated in [125]. They captured a single electromagnetic trace from the encoder's phase modulator and were able to classify all subtraces into one of four classes corresponding to each phase applied to the states. Thus, by identifying all prepared states, they devised an attack able to obtain the full raw key without being detected. No countermeasures were provided. However, we suggest using a shielding material of low permeability to reduce electromagnetic radiation and an electromagnetic jammer to saturate any probe with noise.

Electromagnetic-radiation attack on state measurement In [19], they were able to detect radio-wave signals radiated from avalanche-photodiode detectors after detection events. By first analysing the specific signature of such signals with a source similar to the one used in the implementation, Eve can identify which detector registered a photon during the execution of the protocol, thus obtaining the raw key. The same countermeasures for the previous attack can be applied to detectors. Moreover, placing the detectors as close to each other as possible also increases the interference of radiated electromagnetic signals, decreasing their signal-to-noise ratio.

Timing attack on sifting Some QKD implementations distinguish the signal photons from background noise by also measuring the arrival time of photons on the detectors. This timing information is revealed during public communication for sifting purposes. In [15], by analysing the time distributions of detection events on a setup with four avalanche-photodiode detectors, they were able to identify different centroids for all detectors, with separations on the order of 10 to 100 ps. These were enough for Eve to infer a reasonable portion of the secret key. This side channel can be easily addressed by careful calibration of detection times and by truncating the reported time values to a precision which does not disclose more than enough information to distinguish signal from background noise. The remaining leakage should then be taken into account in privacy amplification.

Power-consumption attack on error correction There are also two main categories of power-consumption attacks, simple power analysis (SPA) and differential power analysis (DPA) [10], analogous to SEMA and DEMA. The same group of the electromagnetic-radiation attack in [18] also analysed the power consumption of the error-correction step of the same plug-&-play implementation [16]. The error correction was done with LDPC codes. They captured a single power trace of Alice's system during her syndrome computation. With SPA, they classified every subtrace into one of two classes corresponding to a computation resulting in a bit 0 or 1. Since the parity-check matrix and the syndrome are known, they were able to reconstruct the raw key. We note that, even if the syndrome was not public, this attack would be able to reconstruct it too. Since Alice's raw key is equal to the corrected key, they only needed to perform the same privacy amplification as Alice and Bob to obtain the secret key. This attack is valid for any implementation using LDPC codes. Their proposed countermeasure is to apply a secret random permutation on the rows of the parity-check matrix to scramble the bits of the syndrome. However, we note that Eve could still obtain some information from this attack. She would know the number of zeros and ones of the syndrome, m_0 and m_1, respectively. Thus, for a syndrome of length m, instead of having to guess the syndrome from a set of 2^m possibilities, she would reduce that set to $\binom{m}{m_0} = \frac{m!}{m_0!m_1!}$ possibilities. This should be taken into account in privacy amplification.

Later, the same group extended the attack to more general error-correction algorithms which compute the parity of bit sequences, such as the cascade protocol, the Winnow protocol and LDPC codes [17]. Using a single power trace with a high signal-to-noise ratio and the public communication of Alice and Bob, they were also able to obtain the full raw key for the cascade and Winnow protocols. Executing these protocols with the public information disclosed by Alice and Bob, they could obtain the corrected key. When they increased the noise present in the power trace, their success rate progressively decreased from 100% to 78.4%. Their proposed countermeasures are to use dedicated circuits to generate noise in parallel to the computation, execute dummy operations during computation to hide the real operations or randomly shuffle the order of operations in parity computations. We note that random shuffling cannot be easily applied to the cascade protocol since the interactive binary search requires the parity computations to be performed in a specific order.

Cache attack on error correction Caches are small memories in processing units used to store information for faster access. In [20], they developed a program-analysis software to analyse the cache state of cryptographic software and upper bound the cache leakage. They used it to test the postprocessing software of an implementation of the BB84 protocol from [126]. Its error-correction protocol uses LDPC codes. A significant cache leakage in the encoding step of error correction was found. With it, they were able to reconstruct the entire raw key. Their proposed countermeasure is a hybrid of software rewriting and increased privacy amplification to minimize the key rate decrease.

Cache side channel in privacy amplification The same group of the previous cache attack had previously used another program-analysis software to analyse the cache state of a simplified version of the privacy-amplification software of the same implementation [127]. This privacy-amplification software uses Toeplitz matrices. They found that the cache state leaked no bits correlated to the secret key, so the software is resistant to cache attacks.

We note that, while most articles on MDI- and DI-QKD argue that they remove most or all side channels from an implementation, they are usually referring to quantum side channels. In fact, we claim that these protocols may still be exposed to classical side channels.

In order to contribute to the research on classical-side-channel attacks on QKD implementations, we will analyse simulated power traces from a privacy-amplification protocol, based on matrix hashing, using machine-learning techniques.

2.4 Machine learning

Machine learning is a subfield of artificial intelligence which provides an automated way of doing data analysis. It encompasses a set of methods which automatically detect patterns in data and use them to predict new data or make decisions involving uncertainty [128].

There are three main approaches to machine learning which depend on the data available [128]: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the models receive a dataset with inputs and their desired outputs and their goal is to learn a mapping from inputs to outputs. If the output variable, usually called target variable, is categorical or nominal, being described by a finite set of values, the problem is known as classification. On the other side, if the output variable is continuous, the problem is known as regression. In unsupervised learning, the models receive only a dataset with inputs and no outputs and their goal is to find possibly-hidden patterns in their inputs. Another less common approach, which falls between the last two, is semi-supervised learning. In it, only part of the input data has an associated output. This approach is useful when it is not feasible to acquire a large-enough dataset of output-labeled data. Finally, in reinforcement learning, the models learn which decisions to make based on reward and punishment signals, also called positive and negative feedback. Instead of receiving an input dataset, the models receive data by interacting with an environment and their goal is to maximize their positive feedback. In our analysis, we will deal with a supervised classification problem.

2.4.1 Data in supervised learning

In a supervised-learning problem, the data consists of multiple individual samples, commonly called records [129]. Usually, records are a set of variables, also called features. A dataset consisting of N records, each with m features, is represented by a $N \times m$ matrix. At least one of the m columns is considered a target feature. There are other types of records, such as images or text, but we will not deal with them in our analysis.

There are multiple approaches when it comes to training the models [128]. The simplest approach is splitting the records into training, validation and testing datasets. The training dataset is used to train the models. Usually, multiple models are trained. Then, the validation dataset is used to evaluate those models in order to choose the best model. Finally, the testing dataset is used to evaluate the chosen model in order to measure its generalization ability, i.e., its performance when predicting outputs for previously-unseen data. One should not use the validation dataset directly for measuring the generalization performance of a model since we already use it to choose a model based on its performance on that data compared to other models. Doing so would introduce a bias in the analysis, possibly leading to an overestimation of the model's general performance [128]. This is a type of information leakage, which generally occurs when we use data already used in the training process for testing the model's performance, or when we use data in the training process which are generally not available at the time of prediction of unseen data. Information leakage generally leads to an overestimation of the generalization performance, and may even lead to choosing a suboptimal model due to decision bias.

Another approach to model training is to split the records into a training dataset and a testing dataset, and use the training dataset in a process called cross validation, which is described in section 2.4.4. This approach is more computationally expensive. Nevertheless, it is the recommended approach for most problems since it allows us to use more data for training, which tends to result in a better-performing model. Therefore, we chose this approach.

2.4.2 Preprocessing data

Before training the models, the data is usually preprocessed [128]. Preprocessing consists not only in splitting the records, but also in removing redundant or irrelevant features and even errors. It is also crucial to remove inconsistencies in the data.

A common way to split independent and identically-distributed data is to use a stratified random sampler. This sampler splits the records randomly into sets containing approximately the same percentage of records, for each value of the target feature, as the full dataset. Therefore, the training and testing datasets are chosen without bias while any imbalance in the target-feature distribution is preserved.

Of the multiple possible transformations, we will only resort to scaling the data. The most common scaling transformations include [129]:

- Scaling all features individually to a certain range, such as $[0, 1]$;

- Standardizing all features individually, i.e., centering them by subtracting their mean and scaling them by dividing by their standard deviation. Thus, standardizing is scaling the data to have mean 0 and variance 1;

- Scaling all features individually by subtracting their median and dividing by the range between the first and the third quartiles, also called the interquartile range. Then, all features will have median 0 and interquartile range 1.

We note that the first scaling may be a more robust transformation for features with very small standard deviations, since it prevents values from becoming very large. On the other side, using the median and interquartile range, instead of the mean and standard deviation, prevents extreme values, also called outliers, from distorting the remaining values excessively, since the former metrics are more robust to outliers than the latter.

We also note that, while these transformations are worthless for some models, such as tree-based algorithms, which are based on variable partitioning and do not consider the variable magnitude, they can be crucial for algorithms which depend on the variable distribution or on the relative scale between variables, such as the logistic regression, the k-nearest-neighbours (kNN) algorithm and the support-vector machine (SVM) [130]. All these models will be briefly described below, with more in-depth mathematical descriptions provided in appendix A.

2.4.3 Models

Multiple models may be used in a classification problem. These models contain two types of parameters [131]. We call parameters to the variables which are iteratively and automatically adjusted during training so that the models improve their predictive ability, and hyperparameters to the variables which we manually adjust before training each model. The hyperparameter space is composed of all possible combinations of hyperparameters for each model. We usually refer to a model with a different combination of hyperparameters as a different model. In practice, we test a finite number of points in the hyperparameter space and use a validation dataset to choose the best one for each model [131].

Below, we briefly describe the models used in our analysis. We refer the interested reader to appendix A, where we provide a mathematical description of these models in a reasonably self-contained way, along with explanations of their respective hyperparameters.

Logistic regression The logistic regression [128], also known as logit regression, is a generalized linear model used for classification, despite its name. Given a set of features and a corresponding target, the logistic regression tries to predict the probability of the target values based on the features. It bases its predictions on the logistic function

$$\sigma(x) = \frac{1}{1+\exp(-x)}, \tag{2.30}$$

using as its argument the scalar product of the vector of input features with a weight vector which describes how each feature influences the predictions.

If our goal is to obtain a target-value prediction instead of a probability, we simply define a threshold probability, such as 0.5, above which we consider the predicted target to be 1, and below which we consider it to be 0 [130].

***k*-nearest neighbours (*k*NN)** The kNN algorithm is a non-parametric supervised-learning model [132]. By non-parametric, we mean there is no set of parameters which summarizes the training procedure like the final weights in a logistic regression. Instead, the model learning is based on memorization of the training records, with a prediction obtained by comparing the new record with the memorized records. Therefore, the cost of classifying new records can be higher than in parametric models, since most computation takes place at the time of classification.

When classifying a new record, kNN finds the k training records closest to that record, according to a specified distance metric, and classifies the new target according to the targets of those k records. The number of neighbours, k, is user-specified. A large k reduces the effects of noise, but makes the classification boundaries less distinct.

The relative scale between the features dictates which features dominate the classification process [130]. Therefore, as mentioned in section 2.4.2, it is crucial to preprocess the features by rescaling them to prevent irrelevant features from dominating the model.

For high-dimensional input spaces, i.e., as the number of features increases, kNN's performance tends to deteriorate quickly [128]. This is because, generally, only a few of those features will be relevant to describing the target. Therefore, the distance between records becomes dominated by the large number of irrelevant features, turning into an unreliable metric.

Even though kNN is a simple model, with a good distance metric and enough training data, it can work quite well. In fact, it was shown in [133] that, as the number of training records $N \to \infty$, the error rate of a kNN with $k = 1$ is never more than twice the optimal error rate.

Naïve Bayes Naïve Bayes methods [128] are a family of supervised-learning algorithms which "naively" assume conditional independence between every pair of input features, given the target value. They use this assumption to simplify the computation of the probability of the target values based on the features using Bayes' theorem as

$$P(y|\boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x}|y)}{P(\boldsymbol{x})} \quad \equiv \quad \text{Posterior} = \frac{\text{Prior} \times \text{Likelihood}}{\text{Evidence}}. \tag{2.31}$$

Different variations of naïve Bayes models may be used depending on the nature of the input features [128]. These variations differ on the assumptions made about the distribution of the likelihood. For continuous features, it is usual to assume the likelihood to be a product of Gaussian distributions and the model is known as Gaussian naïve Bayes. For discrete features, it is common to use a multinomial naïve Bayes, which assumes a multinomial likelihood distribution. This model is most-commonly used for tasks such as text classification, where each feature is the number of times a specific word appears in a text. However, its implementations also work with continuous variables [130]. As shown in appendix A.3, it is interesting note that the multinomial naïve Bayes becomes a linear model when we want to find the logarithm of the posterior probabilities. Finally, when dealing with purely categorical features, the categorical naïve Bayes is commonly chosen. This model assumes the likelihood to be a product of categorical or multinoulli distributions. The implementations of this algorithm also tend to work with continuous variables [130]. These three models are described mathematically in appendix A.3.

In practice, it might be useful to test all methods for some problems, since the categorical naïve Bayes model can adapt particularly well to some continuous data, for instance [130].

It is worth noting that, even if the naïve Bayes assumption (A.5) is not true, the naïve Bayes approach often produces classifiers which work well even with few training records [134]. This is due to the simplicity of these models, which makes them relatively immune to overfitting the training data. The phenomenon of overfitting is more precisely described in section 2.4.4. Nevertheless, even though these can be decent classifiers, the predicted probabilities themselves are known to be poor estimates and should not be considered seriously [135].

Support-vector machine (SVM) SVMs are another family of supervised-learning algorithms [136]. These are based on Vapnik-Chervonenkis (VC) theory [137], a form of statistical learning theory. The goal of SVMs is to find the optimal hyperplane for separating the records according to their target value. This is done by maximizing the minimum distance between the data points of each target value and the hyperplane. This distance is also known as the margin. In general, the larger the margin, the higher the generalization performance the SVM has.

The data points, of each target value, closer to the optimal hyperplane fully define its position and orientation. These data points are known as support vectors. For a linearly-separable problem, these data points sit on the margin boundaries. A representation of this case is presented in figure 2.3. If the problem is not linearly separable, the support vectors are the data points on or within the margin boundaries or on the wrong side of the hyperplane. Note that only the support vectors are needed to build the SVM model, thus making it a memory-efficient model [128].

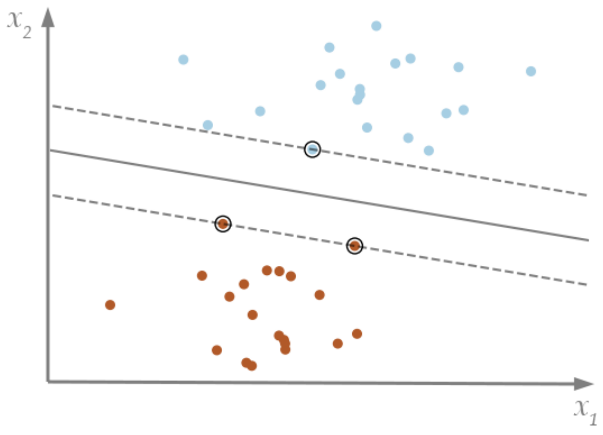


Figure 2.3: Example of the optimal hyperplane for a linearly-separable binary problem. The marked data points are the support vectors.

In addition to linear classification, SVMs are also able to efficiently perform non-linear classification by implicitly mapping the input features into higher-dimensional spaces. This is was proposed in [136] and is done by applying a technique known as the kernel trick. In theory, we first transform the input-feature space and then apply the linear SVM algorithm. This creates a linear model in the transformed input-feature space, but a non-linear one in the original input-feature space.

The feature mapping is defined by a kernel function, which takes the form

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}'), \tag{2.32}$$

where its arguments are vectors of input features and $\phi(\boldsymbol{x})$ is a specific feature-space mapping.

Commonly-used non-linear kernel functions are the gaussian radial-basis function (RBF), polynomial functions and the sigmoid function [128], all defined in appendix A.4. If we set $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \cdot \boldsymbol{x}'$, we obtain the linear SVM [128].

Since SVMs do not scale well to large training datasets, in order to speed up computation, it is common to use kernel-approximation techniques, such as the Nyström method [138]. These techniques numerically approximate the feature-space mapping $\phi(\boldsymbol{x})$ corresponding to a particular kernel function.

As is the case with kNN, SVMs are not scale invariant. Thus, it is highly recommended to scale the input features with one of the methods presented in section 2.4.2 to obtain a good performance [130].

Although SVMs do not directly provide probability estimates, these can be computed with techniques such as Platt scaling [139], in which a logistic regression is trained with the SVM's target predictions using a technique named cross validation, which is described in section 2.4.4. However, this approach can be much more computationally expensive than the non-probabilistic one.

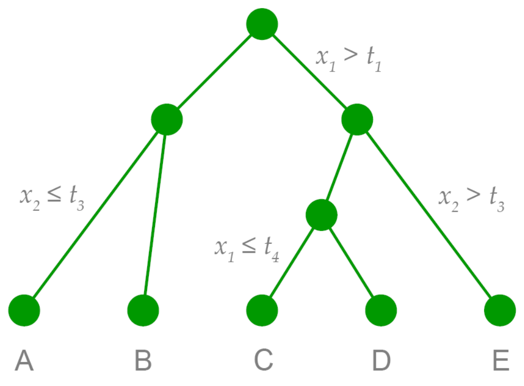
Despite requiring considerable computational resources for high-dimensional spaces, SVMs are known to be effective even when the number of features is greater than the number of training records [131]. Nevertheless, a proper choice of kernel function and hyperparameters is crucial to prevent overfitting of the model to the training data.

Decision tree Decision trees [128] are non-parametric models used in supervised learning. These models recursively partition the input-feature space with axis-parallel splits, i.e, one feature at a time, and assign either a target value or a probability to each subregion.

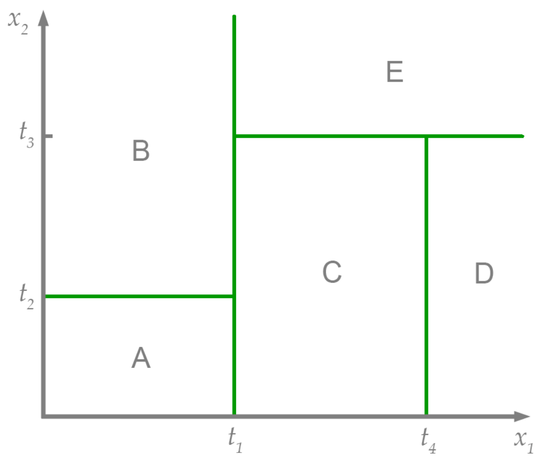
However, finding the optimal partitioning of the input-feature space is a NP-complete problem [140]. Thus, although the correctness of a solution can be verified quickly, in polynomial time, finding a solution requires a brute-force search. This can quickly become unfeasible for high-dimensional feature spaces. Therefore, this partitioning is usually done with a greedy algorithm, which performs locally-optimal splits. Even though this algorithm cannot guarantee a globally-optimal solution, it finds a locally-optimal solution in a reasonable amount of time.

It starts by choosing the input feature and splitting threshold which optimally divide the set of training records into two subsets, according to their target value. This first decision can be represented in a diagram by a decision node, known as the root node, and the chosen feature is the most informative input feature for describing the target variable. The two resulting subsets are then divided recursively, using the same procedure, until the final subsets fulfil certain stopping criteria. Each decision may be represented by a decision node branching off from the previous decision node, and the final subsets are described by nodes, known as leaves, branching off from the last decisions. The output prediction for a new record may be obtained by starting from the root node and traversing the decision-tree diagram according to each decision criterion.

Since decision trees correspond to sequences of binary decisions applied to individual input features, they can be easily visualized and are therefore easy to interpret. An example of a decision tree for a problem in 2 dimensions is presented in figures 2.4a and 2.4b.



(a) Decision-tree diagram.



(b) Input-feature space partitioned by decision tree.

Figure 2.4: Example of decision-tree model for a 2-dimensional problem.

The prediction value of a classification decision tree is either the most common target value of the records in each leaf or the probability of a record classified into a leaf having a specific target value [130]. In regression decision trees, the prediction value is usually the mean or the median target value of the records in each leaf.

Decision trees are very prone to overfitting the training data in high-dimensional spaces when we have few training records [130]. Therefore, it is common to use stopping criteria to improve a decision tree's generalization ability. These are also called pre-pruning strategies [130]. A common one is to allow the tree to grow only to a user-specified depth. The depth of a decision tree is defined as the number of nodes in the longest one-way traversal of the tree from the root node to a leaf. Other common pre-pruning strategies are limiting the total number of leaves, defining the minimum number of training records required to be at a leaf or to be at a decision node for it to be split, and defining the minimum reduction in impurity allowed for a split to happen. The impurity of a set refers to the target-value proportions of its records. A set of records with the same target value is said to be pure. The reduction in impurity associated with a split is mathematically defined in appendix A.5. Another strategy to improve the generalization performance of a decision tree is to introduce randomness into the training process by considering only a random subset of the input features when looking for the best split in each node.

Setting strict values for the pre-pruning strategies mentioned above may also lead to very sub-optimal models. For instance, setting a large minimum reduction in impurity for a split to happen can be counterproductive since it is often found empirically [131] that some optimal splits do not lead to significant impurity decreases, but following splits do. For this reason, it is usual to use more tolerant pre-pruning strategies and, at the end, employ a post-pruning strategy [131]. This may be done by testing all decision-tree variations obtained by removing a subset of the internal decision nodes from the trained tree, and choosing the decision tree with the best generalization performance, for instance.

One advantage of decision trees is they require little data preprocessing, not needing any feature scaling, for instance. Moreover, they are robust to irrelevant features, since only the most informative features will be selected for a split. In fact, the hierarchy of decisions reflects the ability of the features to explain the target variable, with the features associated with splits closer to the root node having greater importance.

However, decision trees are known to be highly unstable, with a small change in the training data potentially resulting in a completely different tree structure and, therefore, final predictions [128].

Random forest One way of reducing the instability of decision trees is by using them in ensemble methods. These combine the predictions of multiple base models to improve that base model's generalization ability [128].

One such method is bootstrap aggregation, also known as bagging and proposed in [141]. In it, a user-defined number of base models are trained with bootstrapped samples, i.e., a subset of the training dataset randomly drawn with replacement. This introduction of randomness helps build a model with a higher generalization performance. The final prediction of a bagging model is either given by a majority voting or an average of the outputs of the base models. When the base model is a decision tree, the ensemble method is known as random forest [142].

It is important to note that simply using bootstrapped training samples tends to create highly-correlated models, since we draw the samples from the same training dataset. This limits the possible increase in generalization performance. We can try to decorrelate the base models by also randomly selecting a subset of the input features for training each model. This helps increase the bagging model's generalization ability [128]. Random forests have the advantage of introducing this additional randomness in the training process by default, since decision trees usually only consider a random subset of the input features in each split, as mentioned above.

The performance of a random forest typically increases with the number of decision trees until it stabilizes [143]. Most often, there is a threshold number of trees after which there is no significant performance gain unless there is a large amount of computational resources available, such as a supercomputer. In [143], the analysis of 29 real medical-domain datasets showed no significant performance gain above 128 trees. In [144], it was shown there are very particular cases where the performance of a random forest peaks and then decreases with the further increase in the number of trees. Nevertheless, the results of their analysis of 306 datasets from the public database OpenML were in agreement with the findings in [143], with the largest performance gain achieved when training the first 100 trees.

By using multiple decision trees, we lose the ability to easily visualize the model. However, feature importance can still be estimated by taking into account the average fraction of records a feature splits when it is used in each base model, as well as the average impurity reduction of those splits [130].

Gradient-boosting machine (GBM) Another ensemble method is boosting [145], in which a user-defined number of models are trained sequentially in such a way that the training of each model focuses more on the training records wrongly predicted by the previous model.

Boosting is particularly effective at creating strong models from a set of weak models [129], i.e., models with a performance only slightly better than the one achieved by random guessing. Hence, the base models for boosting are commonly known as weak learners. In fact, it was proved in [145] that boosting could arbitrarily increase the performance of a weak learner on a training dataset provided the weak learner always performed slightly better than random guessing. Nevertheless, it was later shown that boosting is also effective at increasing the performance of already-strong models [146].

The most popular variant of boosting is adaptive boosting, also known as AdaBoost and proposed in [147]. AdaBoost works by reweighting the training records as the base models are trained. Therefore, this approach requires a model which supports training with weighted records. Usually, the implementations of all models described above support record weights, except for kNN's [130], but decision trees are the most-commonly-used base model for boosting [128]. Record weights are taken into account by multiplying the contribution of a record to a loss function by that record's weight and replacing all statistical measures by weighted measures [131]. Intuitively, when all weights are integers, training a model with weighted records is equivalent to training it with a dataset where each original record has a number of copies specified by its weight.

In AdaBoost, the first model is trained on the original dataset, where every record has the same weight. Then, the weights are readjusted in such a way that wrongly-classified training records are given larger weights than correctly-predicted ones, and a new model is trained with that reweighted dataset. The same reweighting-and-training procedure is done for following models. This way, hard-to-predict records gain increasing influence in the model training as iterations progress.

The record weights are updated according to a user-specified factor $\eta \in (0, 1]$ named learning rate. The smaller the learning rate, the more slowly the weights are updated. This leads to a trade-off between the number of base models and the learning rate [130]. Generally, as we decrease the learning rate, we need to train more models to achieve the same performance.

The final prediction of an AdaBoost model is an average of all base-model predictions weighted according to the number of records wrongly predicted by each model. Models with larger error rates will have a smaller contribution to the final prediction.

In [148], it was observed that boosting may be interpreted as an optimization problem with a particular loss function. Later, [149] expanded on this view by showing how boosting could be extended to use arbitrary differentiable loss functions. This approach became known as GBM.

In a GBM, the base models are regression models even in a classification problem. The most-commonly-used base model in GBM are regression decision trees [128]. Instead of reweighting the training dataset, these models are trained to predict the negative gradient of a loss function for each record. In each training iteration, this loss function takes into account the predictions of the base models trained up to that iteration, weighted in order to minimize the total loss on the training dataset. Using the exponential loss function, defined in appendix A.8, reduces a GBM to the AdaBoost model [128].

The trade-off between the number of base models and the learning rate η present in AdaBoost also exists for GBM [130]. Here, the learning rate controls the contribution of each base model by maintaining or decreasing its effective weight in the ensemble after its respective training iteration. In turn, this affects the weights chosen for the base models trained in following iterations. Empirically, it was found that using small learning rates usually leads to models with higher generalization ability than using $\eta = 1$ [150].

The final prediction of a GBM is a function of a weighted average of the base-model predictions. Depending on the loss function used, the final prediction can either be the predicted target value or the probability of the target being a specific value [128].

In [151], a variation of GBM called stochastic gradient-boosting machine was proposed. It follows the GBM approach but trains each model with a random subset of the training dataset drawn without replacement. As in bagging, this helps increase the generalization performance of the GBM model.

If the base model is a decision tree, feature importance can again be estimated as in random forests, but by instead weighting the averages according to each tree's contribution to the final model [130].

It is important to note that, since ensemble models require training a base model multiple times, they usually demand significantly more computational resources than single models.

2.4.4 Hyperparameter optimization

It is essential to tune the hyperparameters of a model, since a particular choice of these variables may lead to a very different performance on unseen data [131]. This is because the choice of hyperparameters may lead to a model capturing noisy variations in the training data so that it performs better on that data but performs poorly on unseen data. This is the already-mentioned phenomenon called overfitting and leads to poor generalization performance. However, the opposite may also occur. A choice of hyperparameters may lead to a model not capturing enough of the underlying structure in the training data in a way that it performs poorly on both the training data and unseen data. This is called underfitting and it means the model is too simple to properly represent data from that source.

The hyperparameters more closely related to overfitting are those which deal with model regularization [131]. Introducing regularization helps prevent overfitting of a model to training data by reducing the model's complexity, thus improving its generalization ability. An example of the effects of regularization is presented in figure 2.5. Both solution functions represent the given data points with zero loss. However, the regularized solution, in green, likely generalizes better to new data points from the same source.

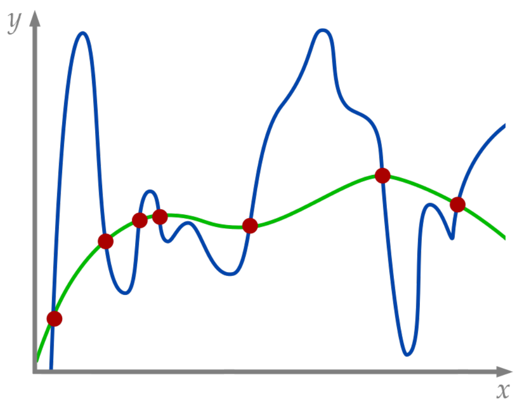


Figure 2.5: Example of the effects of regularization. The blue function is a non-regularized solution to represent the data points. The green function is a regularized solution to the same problem.

Keeping in mind that tuning hyperparameters is crucial to obtain optimal performance, we note that optimal hyperparameter sets can be highly dependent on the data used for training [129].

Cross validation

Cross validation [128] is one of the most common approaches to hyperparameter testing. It is an iterative resampling method which uses different parts of the data to train and validate a model in each iteration. The most-commonly used type of cross validation is k-fold cross validation, whereby the data is divided into k subsets, or folds. The process of splitting the data may also be stratified by using a stratified random sampler. Then, in each iteration, $k-1$ folds are used as the training dataset while the remaining fold is used as the validation dataset. The process is repeated k times, with each of the k folds used exactly once as the validation dataset. The validation results are then combined into a single estimate of the model's performance, most commonly by averaging. As mentioned before, this allows us to use more data for training than simply using a single pair of training and validation datasets, which tends to improve the model's performance. Moreover, by averaging the estimates of a model on k different validation datasets, it helps prevent selection bias by not allowing the characteristics of a particular pair of training and validation sets to affect the performance estimate excessively. Nevertheless, it is more computationally expensive than just using one pair of training and validation datasets since we must train a model k times. A diagram of a 5-fold cross-validation process is shown in figure 2.6.

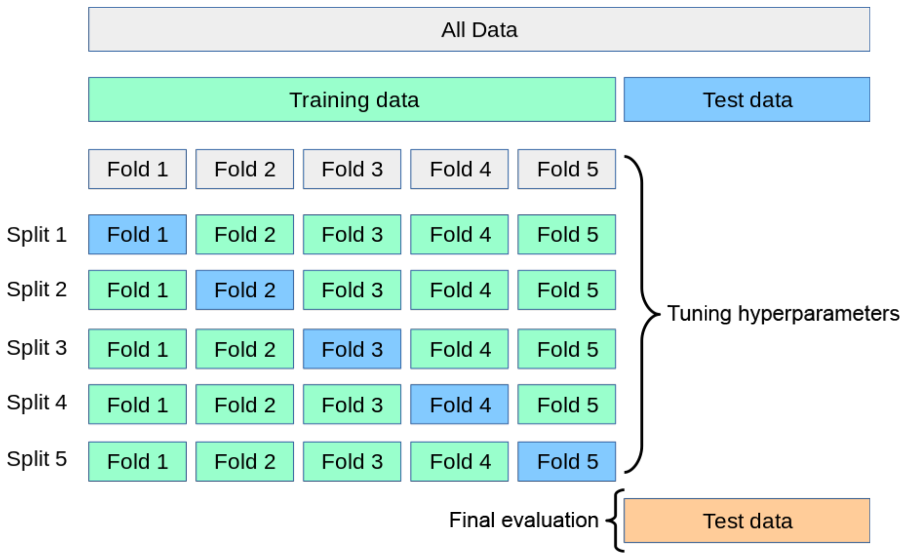


Figure 2.6: 5-fold cross-validation scheme.

It is important to note that the purpose of cross validation is not to build models but to guarantee a fair assessment of a model's generalization performance when selecting the best model. Therefore, after using cross validation to test all sets of hyperparameters, we should retrain the final model with the chosen set of hyperparameters on the full training set.

In order to help select the best hyperparameters for a specific model, we may use hyperparameter-optimization techniques. There are multiple possible approaches, such as grid search, random search, Bayesian optimization, gradient-based techniques and evolutionary-based techniques [152]. Since the most-common approach is to use grid search, we chose this technique.

Grid search

A grid search [152] is a brute-force hyperparameter-optimization approach which consists in exhaustively training models with all combinations of hyperparameters allowed by the user, and choosing the model which performs the best on a validation dataset. A robust way to estimate the models' performance is to use cross validation for each of the models.

Random search

Another very-common way to perform hyperparameter optimization is with a random search [153]. This method replaces the brute-force approach of grid search by randomly selecting points in the allowed hyperparameter space and training models with those combinations of hyperparameters. This method can be a less-computationally-expensive way of finding a well-performing set of hyperparameters. However, it is not guaranteed to be the best-performing set of the allowed hyperparameter space. In the case of continuous hyperparameters, random search can outperform grid search by testing hyperparameter values outside the finite set which would be chosen by a user for performing a grid search.

Lower-fidelity approximation methods

Using a brute-force method like grid search can quickly require tremendous computational resources if we allow large hyperparameter subspaces [152]. In turn, this leads to very long training times. In order to tackle this issue, specially with the need to introduce machine learning into commercial productive environments, research focused on developing model-selection approaches based on approximated performance estimates, which decrease the amount of resources needed to perform hyperparameter optimization.

In [154], a trade-off between the amount of data used to train a model and the fidelity of its performance estimate was explored, with the author noting that testing models on a small subset of the data can be a powerful and computationally-cheap approach to selecting a model. Subsequently-proposed techniques involved iterative elimination schemes which dropped hyperparameter combinations: if they performed poorly on subsets of the data [155]; if they performed poorly on just a few of the cross-validation folds [156]; if they performed worse than a group of top-performing hyperparameter combinations by a determined factor [157]; if they performed worse than the best hyperparameter combination by a user-specified factor [158]; and if an optimistic performance bound for those hyperparameter combinations was still lower than the performance of the best known combination [159]. More recently, a state-of-the-art method, called successive halving and originally introduced in [160], was proposed by [161] for hyperparameter optimization.

Successive halving Successive halving, also referred to as SHA [162], also involves an iterative elimination scheme. It starts the first iteration by training and evaluating models, with the n allowed hyperparameter combinations, in parallel and using only a fraction of a specified resource. This resource is usually the number of records in the training dataset, but may also be the number of estimators in ensemble methods, for instance. Then, according to a user-defined factor f, it keeps only the $\left\lceil \frac{n}{f} \right\rceil$ top-performing hyperparameter combinations for the next iteration. In that iteration, it trains and evaluates the kept models using f times as much of the chosen resource as in the previous iteration, keeping again the $\left\lceil \frac{n}{f} \right\rceil$ best-performing hyperparameter combinations. The iterations continue in this way until either the resource is used in its entirety or there are only f hyperparameter combinations. The selected model is the best-performing one in the last iteration.

Essentially, successive halving is a way of introducing an early-stopping criterion into either grid search or random search. By periodically dropping low-performing hyperparameter combinations, this approach focuses the computational resources on the most promising combinations. It can also be much faster than grid search at testing the same hyperparameter subspace, allowing us to search much larger subspaces with the same computational resources. In [161], successive halving outperformed both a grid search with a uniform allocation of the same quantity of the chosen resource, and other lower-fidelity hyperparameter-optimization methods.

Asynchronous successive halving Successive halving is a sequential algorithm which waits for all models in a iteration to be trained and evaluated to keep the top-performing ones and move on to the next iteration. This allows the possibility for particular hyperparameter configurations to create a large computational bottleneck, thus slowing down the algorithm. In [162], asynchronous successive halving, also referred to as ASHA, was proposed as a massively-parallel framework which removes the need to evaluate and drop low-performing models synchronously, thus further speeding up the computations. In this framework, instead of thinking of sequential iterations, we think of levels with the respective amount of the chosen resource for training the models. Asynchronous successive halving starts by training, in parallel, just enough models from the first level, the one with the smallest amount of the resource, so that we are certain one of the hyperparameter combinations will be kept for the next level. We then prioritize training the second-level model with that hyperparameter set over the remaining first-level models. We continue in this bottom-up approach, prioritizing training models from higher levels and always starting from the upper levels to look for possible hyperparameter-combination promotions to the following levels. We apply the first promotion available and train that model in the next level. If no promotions are available, we train a model with another hyperparameter set in the first level to be able to promote more hyperparameter combinations to the upper levels. By performing all these trainings in parallel, this ensures a near-maximal computational efficiency by fully utilizing the processor resources most of the time.

We note that lower-fidelity methods are not guaranteed to select the best hyperparameter set of the allowed subspace. Instead, they select an almost-optimal set with very high probability and much lower use of computational resources. In order to be able search large hyperparameter subspaces in practical time, for each of the used models, we opted for implementing and using a grid search with asynchronous successive halving in our analysis.

2.4.5 Evaluation metrics

In order to evaluate the models in the multiple steps of training and testing, evaluation metrics are needed. Below, we detail the classification metrics we used.

Accuracy

The target accuracy [129] is the most-commonly used classification metric. It is defined as the fraction of correct predictions. In other words, if we have N records, $\hat{y}^l$ is the predicted output for record l and y^l is its real output, then the accuracy is defined as

$$\text{Accuracy} = \frac{1}{N} \sum_{l=1}^{N} \mathbb{1}\left(\hat{y}^l = y^l\right). \tag{2.33}$$

In case of a dataset with similar proportions for each value of the target feature, a higher accuracy indicates a superior model.

Hamming loss

The Hamming loss [130] is useful when we are dealing with multiple target variables at once. It is defined as the Hamming distance between two sets of variables divided by the cardinality of those sets. Thinking of the sets of variables as strings of symbols, the Hamming distance is the number of positions at which the two strings have different symbols. For example, the Hamming distance between 00111 and 01011 is 2, and the corresponding Hamming loss is $\frac{2}{5}$. Then, the Hamming loss is the fraction of wrongly-predicted outputs.

When evaluating multiple records at once, it is usual to consider the average Hamming loss of the records. More precisely, for N records and m output variables, if $\hat{y}^l_j$ is the predicted value of the j-th output for record l, and y^l_j is the corresponding real output, then the average Hamming loss is defined as

$$\overline{\text{Hamming}} = \frac{1}{N \cdot m} \sum_{l=1}^{N} \sum_{j=0}^{m-1} \mathbb{1}\left(\hat{y}^l_j \neq y^l_j\right). \tag{2.34}$$

Assuming all output variables are equally important, a lower average Hamming loss indicates a better model.

Confusion matrix

In order to further evaluate a classification model, we may use a confusion matrix [129]. If the target variable can take n values, let us assign a number from 0 to $n-1$ to each of them. Then, the confusion matrix can be built by assigning to the entry (i, j) the number of records with real target i and predicted target j. The confusion matrix will thus be a $n \times n$ matrix. In the case of a binary target variable, we usually denote each entry as in figure 2.7.

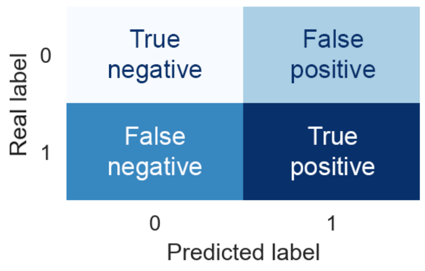


Figure 2.7: Confusion matrix for a binary target variable.

From a confusion matrix, multiple metrics can be easily computed. For example, from figure 2.7, the accuracy may be computed as

$$\text{Accuracy} = \frac{\text{True positives} + \text{True negatives}}{\text{True positives} + \text{True negatives} + \text{False positives} + \text{False negatives}}. \tag{2.35}$$

If we are dealing with multiple target variables, we must build one confusion matrix for each target variable.

2.4.6 Supervised-learning workflow

Let us finally summarize the workflow for this type of machine-learning analysis. In figure 2.8, we present a flowchart of the typical supervised-learning workflow.

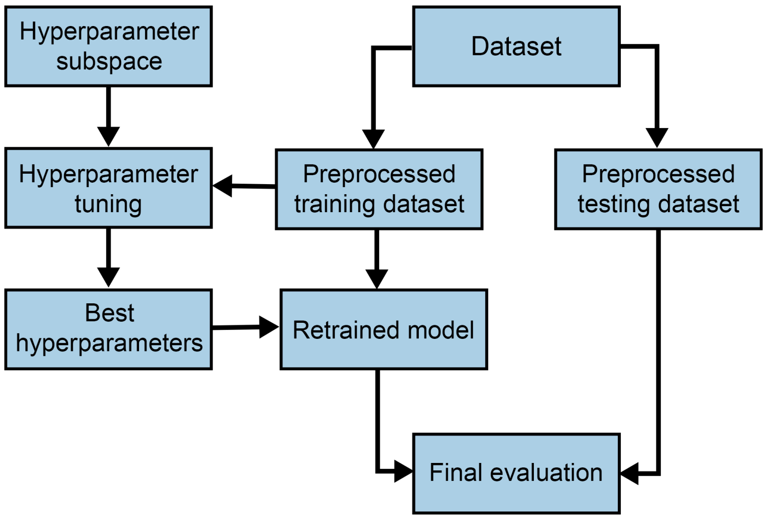


Figure 2.8: Flowchart of the usual supervised-learning workflow.

Firstly, we start by gathering the data and building the full dataset. Then, we split that dataset into a training and a testing dataset. We perform the preprocessing separately on each dataset to prevent any information leakage due to using statistics which depend on data from the other dataset, such as the mean or the median of a feature. We subsequently define the hyperparameter subspace for each kind of model we want to test and perform hyperparameter tuning using the training dataset and a technique described in section 2.4.4, for instance. This will give us the best hyperparameter set for each of the tested models, which we use to retrain that model with the full training dataset. Finally, we evaluate each final model with the testing dataset and a chosen metric to assess its generalization performance.

Chapter 3

Attack methodology

In this chapter, we present our proposed attack on the privacy-amplification step of a general QKD protocol based on matrix hashing, using its power-consumption trace. In section 3.1.1, we detail the implementation considered and how we simulated the power trace. The machine-learning techniques used are specified in section 3.1.2, followed by a description of the attack scenarios and our approaches in section 3.2.

3.1 Proposed attack on privacy amplification

Our proposed attack targets the privacy-amplification step of a general QKD protocol based on matrix hashing. The goal is to estimate how many secret-key bits Eve can obtain by intercepting the power-consumption trace of that step and extracting information from it. In order to perform that extraction, we will use machine-learning techniques instead of the more-commonly-used SPA or DPA, mentioned in section 2.3.1.

At the time of this thesis, there is no physical hardware, similar to that used in laboratory or commercial QKD implementations, available to us, from which we would measure power consumption. Therefore, we will simulate a physical system for privacy amplification. Then, we will run multiple privacy amplification procedures and simulate their power traces so we can perform an analysis based on machine-learning techniques.

To this end, for each scenario, we will first generate:

- a $l \times n$ binary modified Toeplitz matrix, generated by Alice and sent to Bob through the public channel, for extracting a secret key with l bits from a corrected key with n bits;
- 10000 corrected keys with n bits.

Then, we run the privacy-amplification protocol for each corrected key, performing the logical multiplication of the modified Toeplitz matrix and the corrected-key vector and simulating its power consumption in a specific hardware implementation described in the next section.

3.1.1 Power-consumption trace of privacy amplification

In order to properly simulate the power traces of these matrix operations, we will consider what happens at the low hardware level. Firstly, a matrix-vector multiplication with bits can be decomposed into bit multiplications and additions, like in algorithm 1. The power consumed by the initialization of the output vector is redundant to our analysis, so we will ignore it. The remaining operations can be further decomposed into the logic gates which are implemented by the hardware. There are several ways to decompose bit additions and multiplications into logic gates. We will base our decomposition on the logic gates whose power consumption was analysed by [163], namely the NOT, the 2-input NOR and the 2-input NAND gates.

Algorithm 1 Compute logical matrix-vector multiplication.

Input: $l \times n$ matrix $\boldsymbol{M}$, n-vector $\boldsymbol{x}$
Output: l-vector $\boldsymbol{k}$

```
1: procedure MATRIX_VECTOR_MULTIPLICATION($\boldsymbol{M}, \boldsymbol{x}$)
2:     for $i \leftarrow 0$ to $l-1$ do
3:         $k_i \leftarrow 0$                              ▷ Initialization of each entry of the output vector to 0
4:         for $j \leftarrow 0$ to $n-1$ do
5:             $k_i \leftarrow (k_i + M_{ij} \cdot x_j) \mod 2$    ▷ Bit multiplication followed by bit addition. Equivalent to the
                                                          logical scalar product of each row of $\boldsymbol{M}$ with the vector $\boldsymbol{x}$
6:         end for
7:     end for
8:     return $\boldsymbol{k}$
9: end procedure
```

Let us describe the behaviour of these gates before presenting the decomposition. The NOT gate, also called an inverter, takes one binary input and outputs its logical negation. The 2-input NOR gate outputs the negation of the logical disjunction of the two binary inputs, or the negation of a OR gate. The 2-input NAND gate outputs the negation of the logical conjunction of the two binary inputs, or the negation of a AND gate. Both the 2-input NOR and the 2-input NAND gates are functionally complete, meaning only NOR or only NAND gates can be used to generate any other logical function [164]. The truth tables and symbols of these gates are presented in tables 3.1a to 3.1c and figures 3.1a to 3.1c, respectively.

Table 3.1: Truth tables for the gates used.

(a) NOT gate

Input A	Output NOT(A)
0	1
1	0

(b) 2-input NOR gate

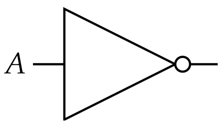
Input A	B	Output NOR(A, B)
0	0	1
0	1	0
1	0	0
1	1	0

(c) 2-input NAND gate

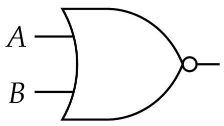
Input A	B	Output NAND(A, B)
0	0	1
0	1	1
1	0	1
1	1	0

Then, the multiplication of two bits, which may be described by an AND gate, can be decomposed into

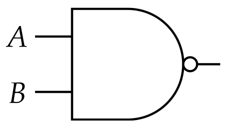
$$A \cdot B : \text{ AND}(A, B) = \text{NOT}(\text{NAND}(A, B)), \tag{3.1}$$



(a) NOT gate.



(b) 2-input NOR gate.



(c) 2-input NAND gate.

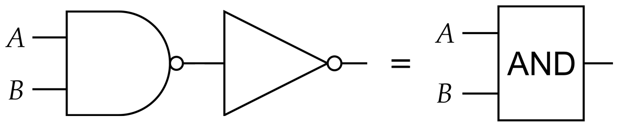
Figure 3.1: Symbols for the gates used.

and the addition of two bits, described by a XOR gate, can be decomposed into

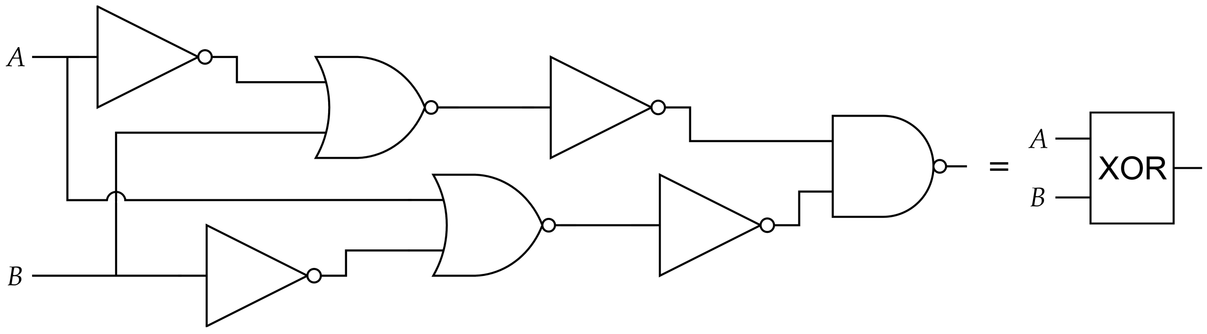
$$\begin{aligned} A + B \mod 2: \ \text{XOR}(A,B) &= \text{NOT}(A \iff B) = \\ &= \text{NOT}(\text{AND}(A \to B, B \to A)) = \\ &= \text{NAND}(A \to B, B \to A) = \\ &= \text{NAND}(\text{OR}(\text{NOT}(A), B), \text{OR}(\text{NOT}(B), A)) = \\ &= \text{NAND}(\text{NOT}(\text{NOR}(\text{NOT}(A), B)), \text{NOT}(\text{NOR}(\text{NOT}(B), A))), \end{aligned} \tag{3.2}$$

where $\iff$ is equivalence and $\to$ is implication. More explicitly, $A \iff B$ is 1 only when A and B are equal, and $A \to B$ is 0 when A is 1 and B is 0, and 1 otherwise.

When processing an addition, there are multiple orders in which the individual gates can be processed. The logic diagrams of these multiplication and addition operations are shown in figures 3.2a and 3.2b, with the chosen gate-processing order made explicit.



(a) Multiplication of 2 bits.



(b) Addition of 2 bits.

Figure 3.2: Logic diagrams of the operations used in the simulated implementation for privacy amplification, time-ordered from left to right.

The power consumption of these gates varies according to their input and to the size of the transistors which make them up, decreasing as the transistors become smaller [163]. We will base the consumption of our simulated implementation on a simplified model of a commercial 0.35-μm-transistor CMOS technology analysed in [163], taking only the dissipated static current into account. This dissipation component gains an increasing importance as the devices become small, specially in the sub-micrometer range, while the other components lose importance [163]. The dissipated static current assumed for the logic gates used is presented in table 3.2, according to the gates' input.

Table 3.2: Static current dissipation of the NOT, 2-input NOR and 2-input NAND gates based on a commercial 0.35-μm-transistor CMOS technology [163].

Gate	Input	Static current (pA)
NOT	0	6.73
	1	7.45
NOR	0 0	13.50
	0 1	12.80
	1 0	7.93
	1 1	5.93
NAND	0 0	5.79
	0 1	7.00
	1 0	11.77
	1 1	14.91

It is clear that processing an addition consumes significantly more power than processing a multiplication. For instance, according to table 3.2, processing $0 \cdot 0$ would dissipate $5.79 + 7.45 = 13.24\, pA$ while processing $0 + 0 \mod 2$ would dissipate $6.73 + 6.73 + 7.93 + 7.93 + 6.73 + 6.73 + 14.91 = 57.69\, pA$.

For simplicity, we will assume each gate takes about the same time to be processed. Then, Eve is able to obtain the power consumption of every gate by measuring the current dissipation at a constant rate, as most devices do, thus acquiring the consumption of a single gate per data point. It is worth noting that considering the power on a shallower level of processing, like the power per addition and per multiplication, as the information present in the data points of a power trace would not be coherent since a multiplication involves 2 gates while an addition involves 7. Therefore, it takes longer for the hardware to process an addition of two bits, creating more data points in the power trace than a multiplication does.

The full logical matrix-vector multiplication is composed by several of these bit multiplications and additions. In our case, there will be n multiplications and n additions per row i, counting the initial addition of the secret-key bit k_i, initialized to 0, and the bit resulting from the multiplication of the first entry of row i of the modified Toeplitz matrix and the first entry of the corrected key. This leads to $l \cdot n$ multiplications and $l \cdot n$ additions in the full matrix-vector multiplication. As with the gates of a single bit addition, there are also multiple orders in which the various multiplications and additions can be processed by the hardware. We will assume an implementation where the operations are computed row by row, and, for each row, all multiplications are computed before the additions. The block diagram of the implementation chosen to process the logical scalar product of each hashing-matrix row with the corrected key is shown in figure 3.3, with the operation-processing order made explicit.

All of these assumptions imply one underlying assumption: Eve must have access to the physical implementation at some point, allowing her to characterize its internal structure and replicate it. We should not discard the possibility of such scenario. In fact, this is specially feasible in QKD networks with remotely-located repeaters, where constant surveillance can be difficult.

We should also note that a regular power trace usually has an overhead contribution from the other processes running on the processor. This simulation assumes a constant overhead, which can be removed from the power trace when processing the data. This may be achieved with DPA or other

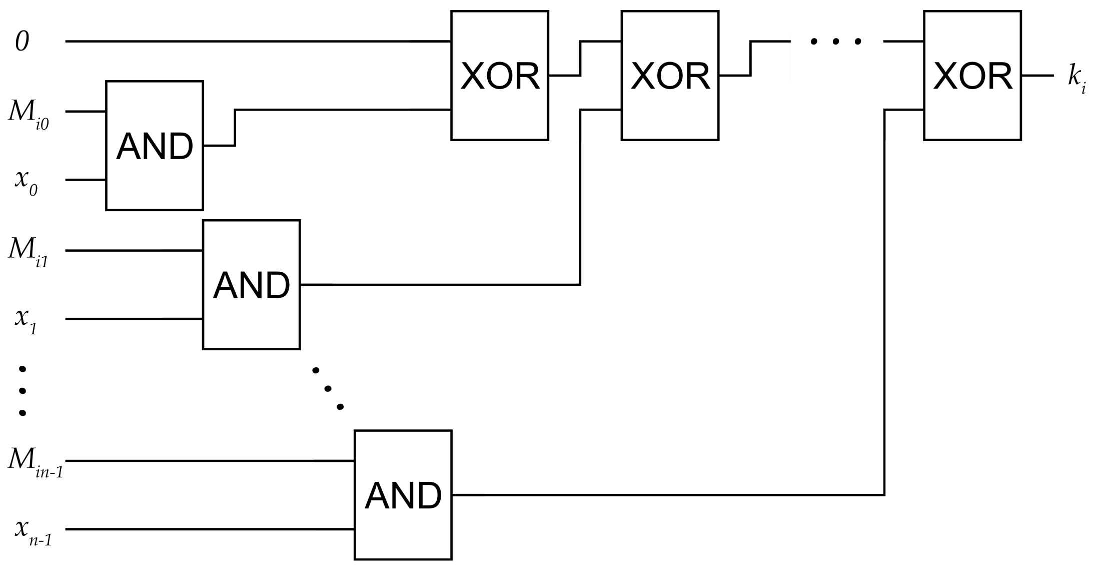


Figure 3.3: Block diagram of the implementation chosen to process the privacy-amplification operations, namely the logical scalar product of each row i of the hashing matrix $\boldsymbol{M}$ with the corrected key $\boldsymbol{x}$, time-ordered from left to right.

statistical techniques, for instance, and is specially feasible when using simpler circuits dedicated to processing the QKD protocol, such as field-programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC), commonly used in real-world QKD implementations [5].

Any real-world measurement is noisy. Therefore, we will also simulate an added noise component in every data point. There are three main types of noise mechanisms in electronic systems [165]: thermal noise, shot noise and $\frac{1}{f}$ noise. Thermal noise is always present at non-zero temperatures due to random thermal vibrations of charge carriers. Shot noise, due to charge carriers crossing a potential barrier, like in semiconductors, is highly dependent on the device type. Moreover, $\frac{1}{f}$ noise, also called flicker noise and due to multiple effects such as property fluctuations of the device materials, depends on the device material, manufacturing quality and semiconductor defects. Of all these, thermal noise is the most encountered. Thermal noise is approximately a white noise, meaning it has similar intensity at different frequencies. Its amplitude follows a Gaussian probability-density function. Thus, it is often modeled as an additive white Gaussian noise, in which the values at any pair of times are identically distributed, statistically independent and, therefore, uncorrelated. In fact, since the central limit theorem indicates the sum of several statistically-independent random processes tends towards a Gaussian distribution, it seems fair to assume most total noise in real-world systems to have such a distribution of instantaneous amplitudes with time. Therefore, we will assume an additive white Gaussian noise of mean zero and standard deviation σ chosen so that the mean absolute value of the noise in all data points ε has appropriate dimensions for our scenarios. We note that this choice is also a common noise distribution considered in template attacks [104]. The standard deviation may be computed by numerically solving

$$\int_{-\infty}^{\infty} \frac{|x|}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = \int_{0}^{\infty} \frac{2x}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = \varepsilon \tag{3.3}$$

with respect to σ.

When discussing our simulation process with academics to gain insights on what to consider for such simulation, it became clear that the sampling rate of the instruments used to acquire side-channel traces is a crucial factor to the success of a such a side-channel attack. Lower-quality instruments usually have signal-sampling rates too low to detect side-channel traces of high-end hardware with enough detail. Moreover, for each specific QKD implementation, there should be a lower bound on the sampling rate below which the information about the secret key present in a side-channel trace is simply not enough to succeed in an attack, even with perfectly-trained machine-learning algorithms. Therefore, we will also consider instruments with different sampling rates. In order to consider values independent of a time scale, we will use sampling intervals, in gates, instead of sampling rates. For instance, we could have an instrument able to capture every gate processed, and another able to capture only one out of every three gates. With an imperfect sampling, another problem arises: the quality of the machine-learning models should depend on the subset of gates sampled. For example, in the case of an instrument with a sampling interval of 3 gates, a model trained with the current dissipation of the first and fourth processed gates can have a very different performance from a model trained on the current dissipated by the second and fifth gates. In our case, we will assume all samplings start with the first gate processed in the protocol, i.e., the NAND gate of the first multiplication involving the first row of the modified Toeplitz matrix. Nevertheless, since the matrix has multiple rows and they are processed sequentially, there is a high probability that samplings of the following rows do not start with the first gate of that row. We just need to make sure the sampling intervals chosen do not divide the length of the corrected keys, n. This introduces enough randomness in the models to mimic the fact that, in a real-world attack, we usually cannot choose which gate to start sampling from, since it is not trivial to synchronize the measurements with the precise start of the QKD procedure.

We will run the simulations in the *Python* language, implementing the following procedure:

1. For each corrected key generated, we will generate the current dissipated by the gates processed in each bit multiplication and addition computed with the implementation shown in figure 3.3, according to table 3.2. For a $l \times n$ matrix, this will generate $9 \cdot l \cdot n$ current values for each corrected key;

2. To each current value, we will add a random noise component sampled from an additive white Gaussian noise model with mean zero and standard deviation appropriately chosen;

3. Then, we will downsample the data according to the sampling interval s of the instrument considered, keeping only the first of every s data points. The discarded data points correspond to the processing of the privacy-amplification operations done during the dead time of the acquisition instrument. Thus, we will have simulated the signal acquisition.

After obtaining the power traces from the considered hardware implementation, we will proceed to the next step of the attack, where we will use machine-learning techniques to try to extract information about the secret key from the power traces.

3.1.2 Machine-learning analysis

We will use the popular *scikit-learn* machine-learning library for *Python* [130], in which some of the machine-learning algorithms used in this analysis are already implemented.

A record is the subset of data corresponding to a generated corrected key. We start by splitting the data into a training and a testing dataset using a stratified random sampler. We will use 80% of the records for training and the other 20% for testing. Each current data point of a power trace will correspond to a particular feature in our models. That is, if we have k data points per corrected key, we will create models with k input features. Then, we define the models we want to use and the hyperparameter subspace we want to optimize our models in. As described in section 2.4.3, we will use the following models:

- Logistic regression;
- k-nearest neighbours (kNN);
- Gaussian naïve Bayes;
- Multinomial naïve Bayes;
- Categorical naïve Bayes;
- Support-vector machine (SVM);
- Decision tree;
- Random forest;
- Stochastic gradient-boosting machine (GBM) with regression decision trees.

Since we will be dealing with evaluation metrics which use predicted target values, we will apply a threshold probability of 0.5 on the models which return probabilistic outputs.

In tables 3.3a to 3.3g, we present the hyperparameter subspace allowed for each model. A clarification of each hyperparameter is presented in appendix A. In the values used for the hyperparameter γ of the SVM kernel, k is the number of input features and σ^2 is the average variance of a training record. Note that we had to reduce the number of hyperparameters for random forest and GBM due to the high time consumption of these algorithms.

Besides optimizing the hyperparameters, we will also optimize the preprocessing done to the data by allowing the following transformations before all models:

- No transformation;
- Scale all features individually to the range $[0, 1]$;
- Standardize all features individually to mean 0 and variance 1;
- Scale all features individually to median 0 and interquartile range 1.

Table 3.3: Hyperparameter subspace allowed for each model.

(a) Logistic regression

Hyperparameter	Values
Solver	L-BFGS, SAGA
Regularization	None, L1, L2
λ	0.01, 0.1, 1, 10, 100

(b) kNN

Hyperparameter	Values
Number of neighbours	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21
Neighbour weight	Uniform, Inverse of distance
Metric	Manhattan, Euclidean, Chebyshev

(c) Naïve Bayes

Hyperparameter	Values
Prior	Uniform, Target-value prior probabilities
α	0, 0.001, 0.01, 0.1, 1, 10, 100, 1000

(d) SVM

Hyperparameter	Values
Kernel	Linear, Gaussian RBF, Third-degree polynomial, Fourth-degree polynomial, Sigmoid
γ	1, $\frac{1}{k}$, $\frac{1}{\sigma^2}$, $\frac{1}{k\sigma^2}$
b	-1, 0, 1
λ	0.01, 0.1, 1, 10, 100

(e) Decision tree

Hyperparameter	Values
Loss function	Gini impurity, binary Shannon entropy
Maximum depth	5, 10, 20, ∞
Maximum percentage of features to consider in each split	10%, 30%, 50%, 70%, 100%
Minimum number of records in a leaf	1, 5, 10, 50, 100
Minimum impurity reduction	0, 10^{-6}, 10^{-4}, 10^{-2}

(f) Random forest

Hyperparameter	Values
Number of trees	243
Loss function	Gini impurity, binary Shannon entropy
Maximum depth of each tree	5, 20, ∞
Maximum percentage of features to consider in each split	10%, 50%, 100%
Minimum number of records in a leaf	1, 10, 100
Minimum impurity reduction	10^{-6}, 10^{-4}, 10^{-2}

(g) GBM

Hyperparameter	Values
Number of trees	243
GBM's loss function	Exponential loss (AdaBoost), Log loss (LogitBoost)
Learning rate	0.01, 0.1, 0.9
Subsample size for each tree	50%, 75%, 100%
Tree's loss function	MSE, MAE
Maximum depth of each tree	5, 20, ∞
Maximum percentage of features to consider in each split	10%, 50%, 100%
Minimum number of records in a leaf	1, 10, 100
Minimum impurity reduction	10^{-6}, 10^{-4}, 10^{-2}

Using these hyperparameter subspaces, we will run a grid search for each model with the training dataset, using stratified 5-fold cross validation and asynchronous successive halving with $f = 3$. While we will use the number of records as the halving resource for most models, in the ensemble methods, namely random forest and GBM, we will use the number of trees as the halving resource instead of tuning it, due to the findings of [143, 144] stated in section 2.4.3. We chose 243 trees since it is the smallest multiple of f greater than 128. Then, there will be 6 halving iterations for ensemble methods, using 1, 3, 9, 27, 81 and 243 trees, respectively. For the other models, we will start with 100 records and increase that number until we reach the limit of 8000 records.

After obtaining the combinations of hyperparameters for which each model performs the best on the validation-set folds, according to a metric we will specify later, we will train the models with those hyperparameters on the full training set. We will then evaluate the trained models with the testing dataset and the chosen metrics and compare the model results to find the best-performing and most-generalizable model.

All the code used for this thesis can be found on GitHub[1], along with comments detailing the nuances of its implementation.

3.2 Attack scenarios and approaches

In order to assess the robustness of this attack, we will use it to try to break privacy-amplification protocols with different hashing-matrix sizes, measurement noise levels and instrument's sampling intervals. We will test the following scenarios:

- Modified-Toeplitz-matrix size: 8×16, 16×32, 32×64;
- Mean absolute value of measurement noise: 0 pA (no noise), 0.1 pA, 1 pA;
- Instrument's sampling interval: 1, 3, 5, 10, 20 and 30 gates.

Note that the noise levels were chosen to be near the order of magnitude of the current dissipation of the gates considered. A high-enough noise level can always hinder the success of such a side-channel attack.

We will also test different variations of the machine-learning analysis to assess the best approach for Eve. We note that the goal of this analysis is not to empower possible eavesdroppers, but to understand the worst-case outcome for Alice and Bob.

First of all, we will assess the ability of these machine-learning models to extract 1 bit of information from the power traces. To this end, our target variable will be a flag which is 1 if the secret key has more ones than zeros, and 0 otherwise. The goal will be to maximize the target accuracy, which will then be used to compare the models. We will refer to these as 1-bit models.

Secondly, we will try to find the full secret key by assigning each of its bits to a different target. We will then have l binary targets, making this a multi-label classification problem. In our case, this can be solved

[1]https://github.com/joaofbravo/Hacking_QKD_with_ML

by training one model for each of the targets. Therefore, it is much more computationally expensive. The goal will be to minimize the average Hamming loss of the predicted keys. We will address each ensemble of l models used to predict one key as full-matrix model, since each individual model of the ensemble receives input data generated from operations involving the full hashing matrix.

In order to try a less time-consuming approach, we will also split the dataset rows, which correspond to full power traces, into subrows corresponding to the power trace of the logical scalar product of each matrix row with the corrected key. Instead of 10000 records, we will have $10000 \cdot l$ records. Since the logical scalar product of a matrix row with the corrected key fully determines one bit of the secret key, we will train models for each of the l secret-key bits with the 10000 subrows which correspond to the operations determining that bit. Thus, the models will use much less features, which is less computationally expensive. This approach also reduces the amount of unnecessary data provided to the models, since the power trace corresponding to the computation of the other $l-1$ bits is independent of the target bit. This may increase the accuracy of predictions. One downside of this approach is that scenarios with sampling intervals larger than 1 gate can lead to subrows with a different number of data points. Specifically, there will be subrows with one less data point. Since the models need records with the same number of variables, we solve this issue by adding a neutral data point of 0 pA at the beginning of those subrows. While we hope the models learn how to differentiate these subrows by their first data point, this can lead to a worse prediction performance in less flexible models. The goal will again be to minimize the average Hamming loss of the predicted keys. We will address each ensemble of l models used to predict one key as row model, since each individual model of the ensemble takes input data generated from operations involving only one row of the hashing matrix.

This analysis is crucial to decide if it is necessary to apply countermeasures, either on the devices used for sharing keys or on the protocol itself. These countermeasures may result in additional steps that ensure the communication is secure, which can reduce the protocol's key rate.

Chapter 4

Results and discussion

In this chapter, we present and analyse our results. In section 4.1, we present and discuss aspects of some of the generated power traces. In section 4.2.1, we analyse the results of the 1-bit-model test case. In sections 4.2.2 and 4.2.3, we do the same for the full-matrix and row models, respectively. In section 4.3, we compare the attack approaches, choosing the best approach depending on the attack scenario. Finally, in section 4.4, we discuss some possible countermeasures to prevent similar side-channel attacks.

4.1 Power traces

From the multiple corrected keys, we generated power traces for each scenario. As an example, we present the power trace of the logical scalar product of the first 8×16-matrix row with one corrected key, using a noise level of 0.1 pA and a sampling interval of 1 gate, in figure 4.1. The initial current peaks should correspond to the processing of the NAND gates of bit multiplications, with the highest peaks corresponding to $1 \cdot 1$. The final higher current peaks should correspond to the last NAND gates processed in each bit addition, while the smaller peaks can either correspond to NAND or NOR gates.

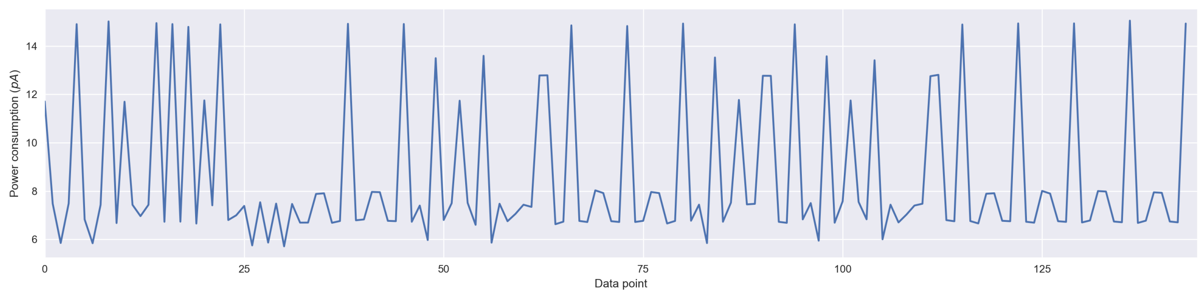


Figure 4.1: Power trace of the logical scalar product of the first hashing-matrix row with a corrected key, for a 8×16 matrix, noise level of 0.1 pA and sampling interval of 1 gate.

By lowering the resolution of the measuring instrument, therefore increasing the sampling interval, Eve will collect fewer data points. In figure 4.2, the same power trace is shown as if it was measured by an instrument with a sampling interval of 3 gates. Notice how a lot of information is lost. For instance, we are only able to identify a few NAND(1, 1) operations.

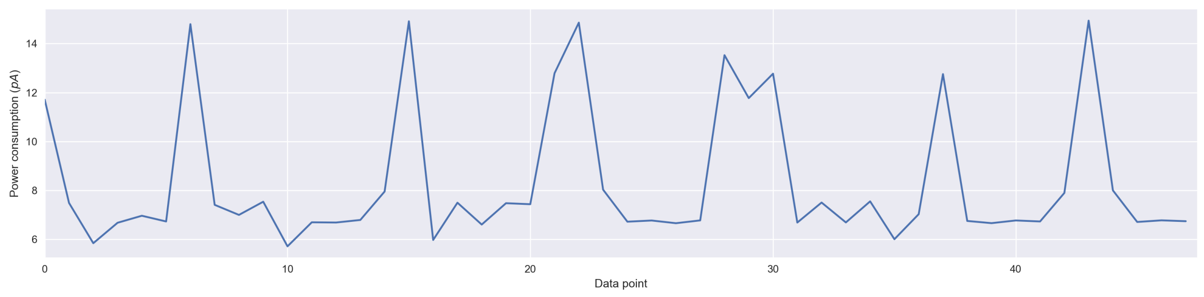


Figure 4.2: Power trace of the logical scalar product of the first hashing-matrix row with a corrected key, for a 8×16 matrix, noise level of 0.1 pA and sampling interval of 3 gates.

A large amount of noise can also destroy a lot of information. In figure 4.3, we present a power trace similar to the one in figure 4.1, but with a measurement noise level of 1 pA. Notice how some peaks were perturbed enough that some NAND(1, 1) operations, clearly identifiable in figure 4.1, now have current dissipation similar to NOR(0, 0) operations, making them indistinguishable.

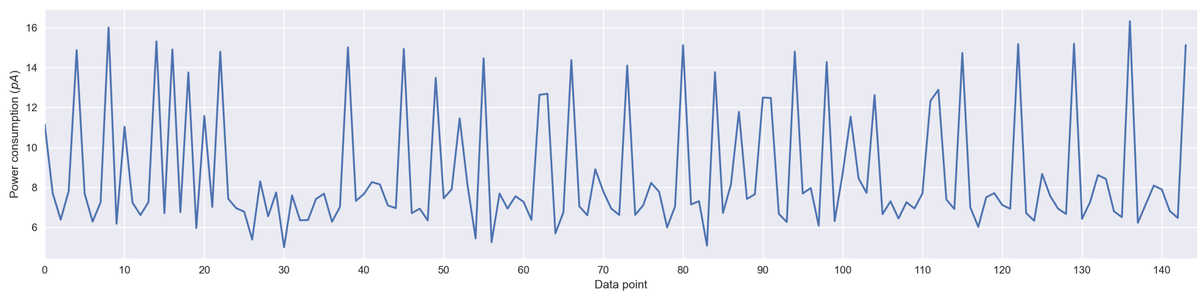


Figure 4.3: Power trace of the logical scalar product of the first hashing-matrix row with a corrected key, for a 8×16 matrix, noise level of 1 pA and sampling interval of 1 gate.

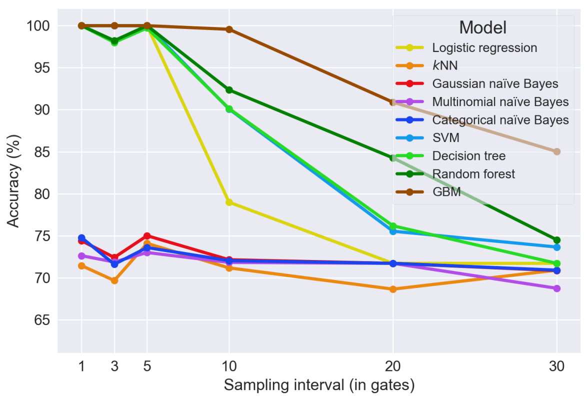
These information losses will affect the performance of the models in each considered scenario.

4.2 Model results

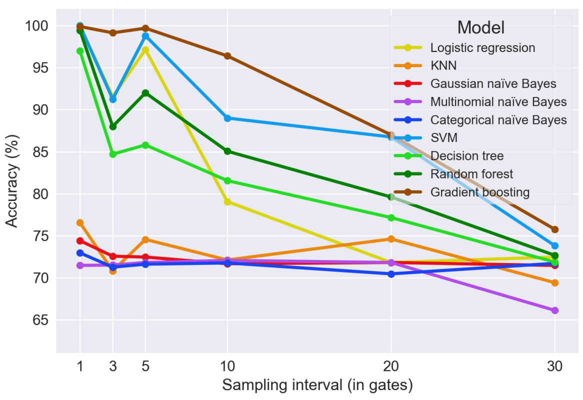
Here, we analyse the results of the models built from the generated power traces for the scenarios and approaches described in section 3.2. A summary of all model results from the three studied approaches can be found in appendix B.

4.2.1 1-bit models

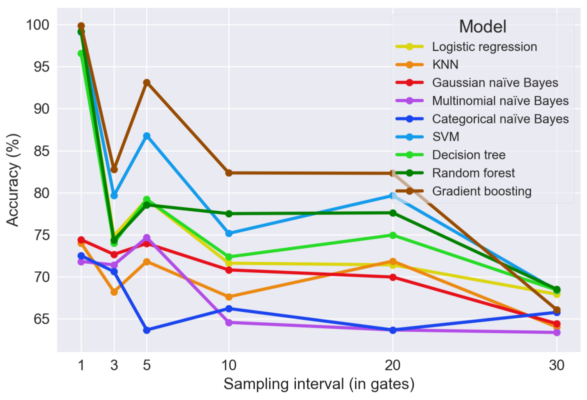
After splitting the data, optimizing the hyperparameters with grid search, retraining each model in each scenario with the best hyperparameters and evaluating the accuracy of the final models, we summarized the results and compared those models. As an example, in figures 4.4a to 4.4c, we show how the target accuracy of the 1-bit models changes with the sampling interval and noise level, for a 8×16 hashing matrix.



(a) For no noise.



(b) For a noise level of 0.1 pA.



(c) For a noise level of 1 pA.

Figure 4.4: Accuracy of 1-bit models by sampling interval, for each noise level, with a 8×16 hashing matrix.

In these scenarios, GBM performed the best, always being able to extract 1 bit of information from the power traces in noiseless environments even when sampling down to 1 out of every 5 gates. The accuracy barely dropped with a sampling interval of 10 gates which may be surprising since each bit of the hashing matrix is directly involved in 9 gate operations, 2 for its multiplication with the corresponding corrected-key bit and 7 for the addition to the corresponding secret-key bit k_i. Then, sampling intervals larger than 9 gates should miss information about some of the bits of the hashing matrix. However, this can be explained by the fact that k_i can be fully determined by multiple small subsets of operations involving row i, according to algorithm 1. For instance, it can be fully determined by the last addition involving row i, or by the second to last addition and the last multiplication, which determine the last addition and therefore k_i, or by the third to last addition and the two last multiplications, and so on.

When introducing a noise level of 0.1 pA, the accuracy of GBM did not drop significantly for sampling intervals up to 5 gates. However, it dropped around 3% for a sampling interval of 10 gates, which indicates that the noise destroyed crucial information present in the gathered data points. However, an accuracy of 96.40% is still high enough to consider the bit extraction successful the vast majority of the time. When introducing a large amount of noise, of the level of 1 pA, the accuracy dropped considerably for sampling intervals larger than 1 gate, since a lot of operations became indistinguishable due to the noise itself.

However, when sampling all gates, the accuracy did not drop because there was enough redundant information to fully identify the target bit, despite the noise destroying part of the information.

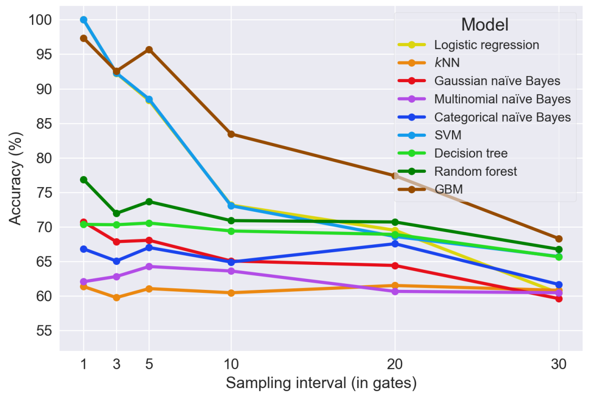
Interestingly, sampling 1 out of every 3 gates usually led to worse results than sampling 1 out of every 5 gates. This is due to the problem addressed in section 3.1.1 about how the performance of a model depends on the subset of gates sampled. The gates sampled in these two scenarios are mostly different, and the subset sampled when measuring 1 out of every 5 gates clearly provided more information about the target bit.

When sampling 1 out of every 20 or 30 gates, the accuracy dropped considerably since there was not enough information to determine the target bit, even in a noiseless environment. Nevertheless, when sampling 1 out of every 20 gates with a noise level of 0.1 pA, we were able to successfully extract 1 bit of information 87% of the time.

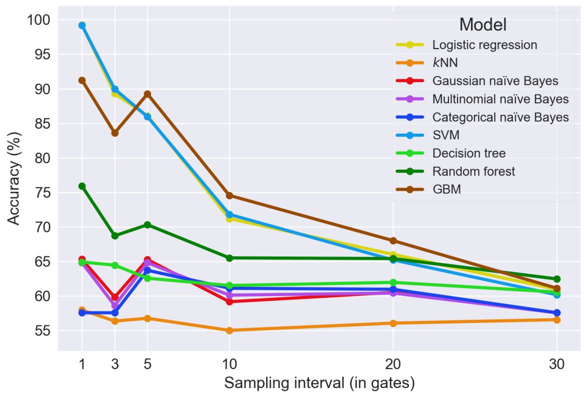
In spite of this, GBM was always able to extract 1 bit of information from the power traces when sampling all gates, independently of the noise level. This makes GBM a promising model for the other approaches.

In these scenarios, SVM, logistic regression, random forest and decision tree all exhibited a worse performance than GBM while still having a decent accuracy for smaller sampling intervals when there was not a large amount of noise in the measurements. kNN and all naïve Bayes showed really poor performance, independently of the sampling interval or noise level. kNN's performance may indicate we have too many irrelevant features, while naïve Bayes' performance may also be attributed to the fact that the features are clearly not independent. For instance, processing $0 \cdot 0$ always generates a data point around 5.79 pA followed by another around 7.45 pA.

Lets us now compare these scenarios with the ones with larger hashing matrices. In figures 4.5a and 4.5b, we present the target accuracy of these models for 16×32 and 32×64 hashing matrices, with a noise level of 0.1 pA. The remaining scenarios, which can be found in appendix B.1, will also be briefly analysed.



(a) For a 16×32 hashing matrix.



(b) For a 32×64 hashing matrix.

Figure 4.5: Accuracy of 1-bit models by sampling interval, for 16×32 and 32×64 hashing matrices, with a noise level of 0.1 pA.

The size of the hashing matrix seems to have a great influence on the ability of the models to extract a single bit, decreasing it as the matrix size gets larger.

Moreover, we note that both logistic regression and SVM performed better than GBM when sampling all gates in these scenarios and when sampling down to 1 of 5 gates in the corresponding noiseless scenarios. Nevertheless, we were always able to extract 1 bit with near-perfect accuracy when sampling all gates, except with a high noise level of 1 pA. For sampling intervals of 5 gates or more and any noise, GBM was again the better performing model.

In the scenarios of figures 4.5a and 4.5b, the best accuracy attained for a sampling interval of 3 gates decreased from 99.15% to 92.55% and to 89.30% when using the larger-size hashing matrices. For a sampling interval of 5 gates, the best accuracy decreased from 99.70% to 95.70% and to 89.25% for the larger hashing matrices.

For the larger hashing matrices, only GBM, SVM and logistic regression exhibited a decent performance, while the other models performed much worse, almost independently of the sampling interval. Random forest and decision tree experienced the highest degradation in performance with the increase of the hashing-matrix size. The worst results for each model were generally obtained for the scenario with a 32×64 hashing matrix and a noise level of 1 pA, as expected.

We note that some models showed a slightly higher accuracy with larger matrices. This can be due to multiples causes, namely the stochastic nature of the model training for some of the models, the different randomly-generated corrected keys for each hashing-matrix size, and the randomly-generated hashing matrices themselves which, despite having been generated to have around half ones and half zeros, may have slight biases towards a certain resulting bit in some of their rows.

We can further analyse each model by looking at the confusion matrix of its predictions. For instance, with the confusion matrix of the logistic regression for the scenario with a 32×64 hashing matrix, noise level of 0.1 pA and sampling interval of 3 gates, shown in figure 4.6, we can conclude there were 901 true zeros, 885 true ones, 117 false zeros and 97 false ones. We can also conclude that there were more zeros predicted by the logistic regression, which indicates it is a slightly-biased model.

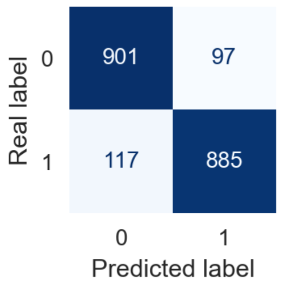
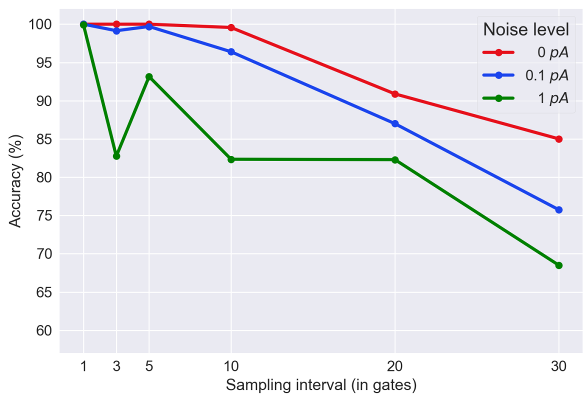
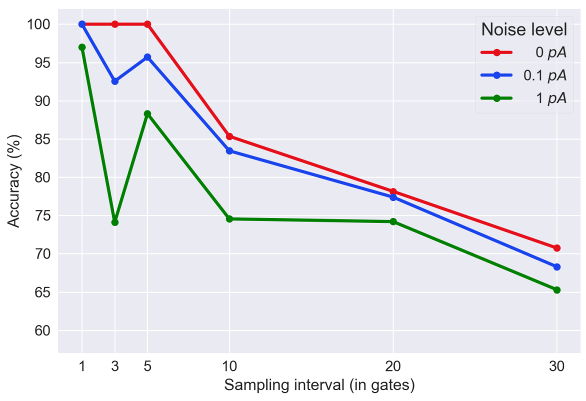


Figure 4.6: Confusion matrix for a 1-bit logistic regression trained on the data acquired with a 32×64 hashing matrix, noise level of 0.1 pA and sampling interval of 3 gates.

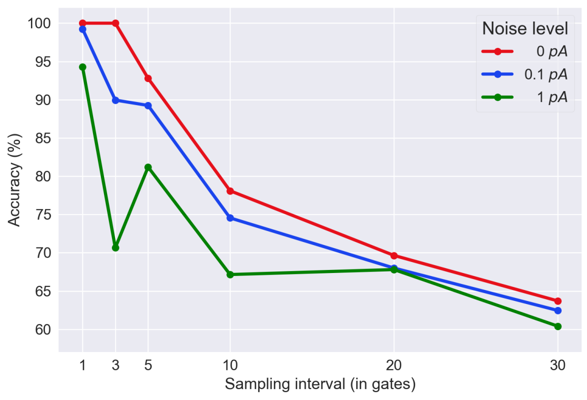
After performing such a study, Eve could then decide to use the best model according to the particular scenario of the attack. Let us then summarize the accuracy of the best 1-bit models for each scenario, by sampling interval, and show it in figures 4.7a to 4.7c.



(a) For a 8×16 hashing matrix.



(b) For a 16×32 hashing matrix.



(c) For a 32×64 hashing matrix.

Figure 4.7: Accuracy of the best 1-bit model by sampling interval, for each noise level and hashing-matrix size.

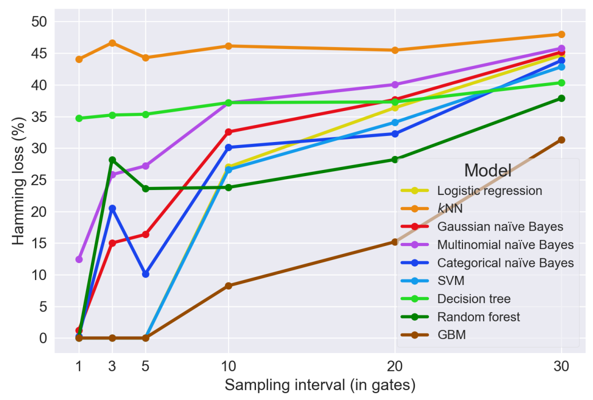
It is now evident how increasing the noise and the hashing-matrix size degrades the performance of 1-bit models. However, we note that there was no significant degradation when sampling all gates if the noise level was around or below 0.1 pA and also when sampling 1 out of 3 gates in noiseless environments. Again, the scenario with a 32×64 hashing matrix and a noise level of 1 pA showed the worst results, as expected.

Let us now assess how these models generalize their predictive power to predicting full secret keys.

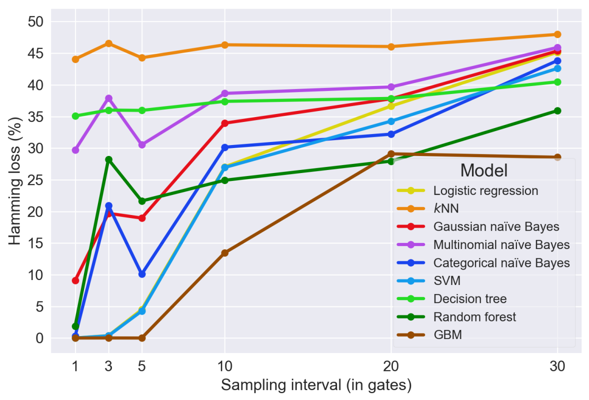
4.2.2 Full-matrix models

We completed the same steps as in the 1-bit case, except we evaluated the performance of the full-matrix models according to the average Hamming loss of their predicted keys. In figures 4.8a to 4.8c, we show their average Hamming loss by sampling interval and noise level, for a 32×64 hashing matrix.

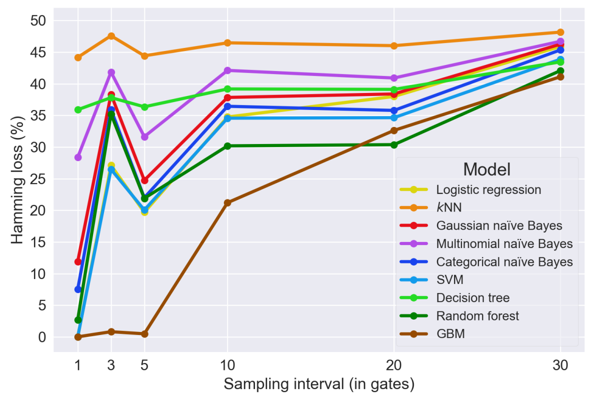
In these scenarios, GBM was again the best-performing model, virtually always being able to predict the full secret key for sampling intervals up to 5 gates and all noise levels. For smaller sampling intervals, SVM and logistic regression showed a similar performance to GBM. However, GBM was the only model capable of consistently predicting the full secret key in very noisy environments, for sampling intervals larger than 1 gate.



(a) For no noise.



(b) For a noise level of 0.1 pA.



(c) For a noise level of 1 pA.

Figure 4.8: Average Hamming loss of the predicted keys of full-matrix models by sampling interval, for each noise level, with a 32×64 hashing matrix.

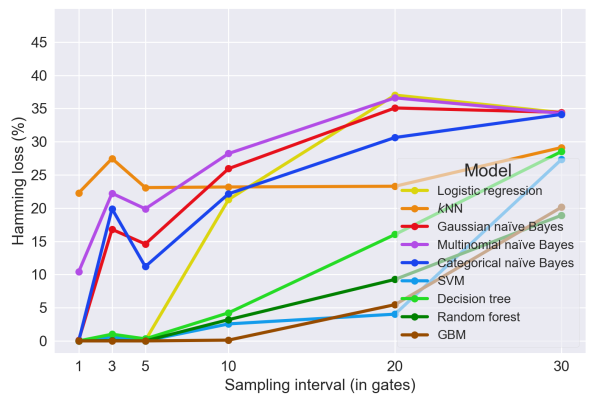
Some of the models showed again the importance of the sampled-gate subset, by performing better with a sampling interval of 5 gates than with one of 3 gates.

When sampling 1 of every 10 gates, GBM predicted 8.27% of the secret-key bits wrongly with no noise, 13.48% with a noise level of 0.1 pA, and 21.22% with a noise level of 1 pA. Again, for sampling intervals of 20 and 30 gates, the information was not nearly enough to predict every bit of the secret key and the performance dropped considerably.

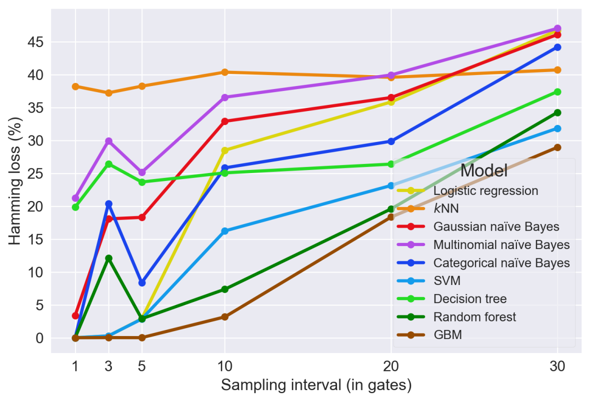
Random forest and the categorical and Gaussian naïve Bayes had a generally-poorer performance, followed by decision tree and the multinomial naïve Bayes, with kNN being the worst. We note that, despite this, some of these models were still able to always predict the full secret key when sampling all gates, for noises up to 0.1 pA, with the categorical naïve Bayes being an unexpected case since it is designed for categorical variables as explained in section 2.4.3.

In order to assess how the hashing-matrix size affects the full-matrix models, we show the average Hamming loss of the predicted keys of these models for 8×16 and 16×32 hashing matrices, with a noise level of 0.1 pA, in figures 4.9a and 4.9b. We also briefly analyse the other scenarios, presented in appendix B.2.

For sampling intervals up to 5 gates, GBM performed virtually flawlessly in every scenario. There was a clear degradation of its performance for sampling intervals above 5 gates. In particular, for a sampling



(a) For a 8×16 hashing matrix.



(b) For a 16×32 hashing matrix.

Figure 4.9: Average Hamming loss of the predicted keys of full-matrix models by sampling interval, for 8×16 and 16×32 hashing matrices, with a noise level of 0.1 pA.

interval of 10 gates, it went from virtually no wrongly-predicted bits to 3.17% and 13.48% of predicted bits incorrect for larger hashing-matrix sizes. Nevertheless, GBM was the best model in virtually every scenario.

We note that decision tree and random forest are very affected by the hashing-matrix size, having the greatest increase in Hamming loss as we increased the matrix size. The categorical naïve Bayes was the least affected by the hashing-matrix size, showing the least change in Hamming loss.

When analysing the confusion matrices of the best models, we reach an interesting conclusion. For instance, let us interpret the confusion matrices of the SVM for the scenario with a 8×16 hashing matrix, noise level of 0.1 pA and sampling interval of 20 gates, shown in figure 4.10.

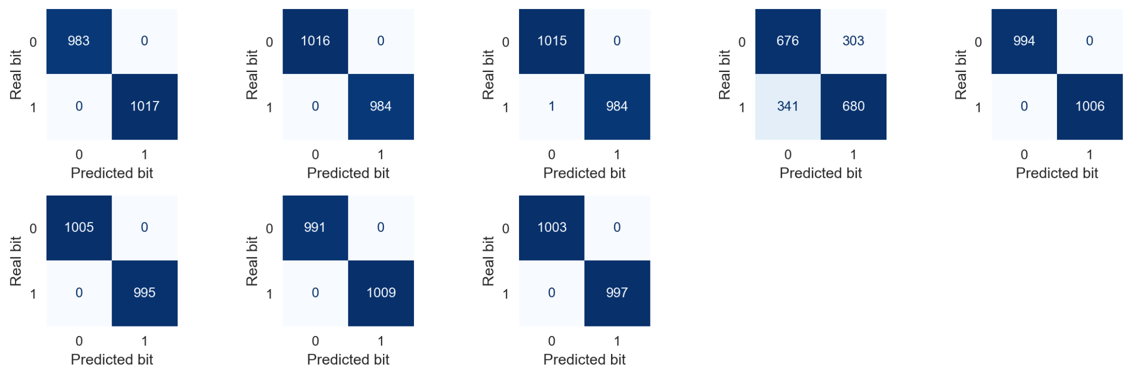


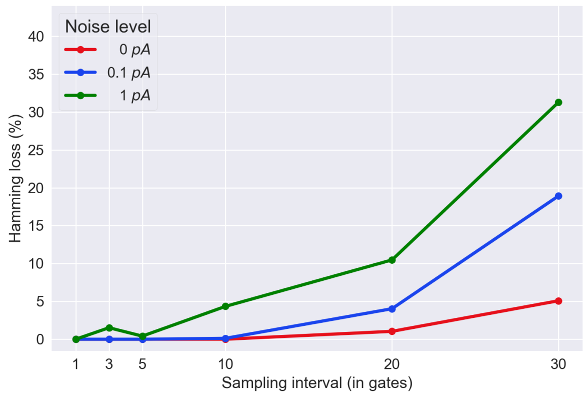
Figure 4.10: Confusion matrices for a full-matrix SVM trained on the data acquired with a 8×16 hashing matrix, noise level of 0.1 pA and sampling interval of 20 gates.

This SVM has an average Hamming loss of 4.03%. We conclude that it can predict 7 of the 8 target bits with virtual certainty, while it struggles when predicting the fourth bit of the secret key. This may be an example of how noise can destroy information about a particular bit due to its stochastic nature.

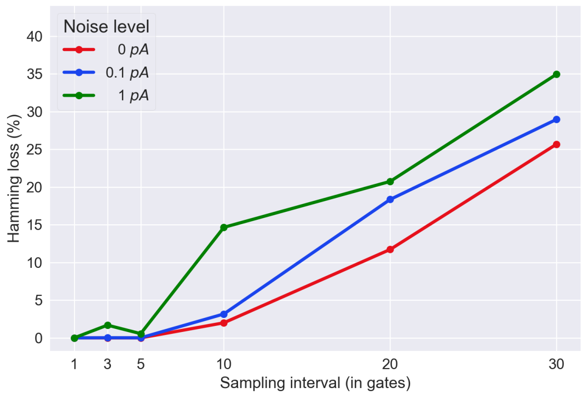
By analysing such metrics, Eve can then know which bits her model can predict with near certainty. Thus, when she tries to decrypt the messages shared between Alice and Bob, she can use brute force and try all variations of the predicted key with a subset from the power set of the hard-to-predict bits inverted. We define the power set as the set of all subsets of those bits, ranging from the empty set to the set of all hard-to-predict bits. For instance, in the scenario above using the SVM, she would try the predicted key $k_0k_1k_2k_3k_4k_5k_6k_7$ and then the key $k_0k_1k_2\neg k_3k_4k_5k_6k_7$, where $\neg k_3$ denotes the inversion of k_3. In general, if Eve's model has difficulty predicting k bits of a l-bit secret key, she will often only need to test 2^k key possibilities at most, instead of 2^l. Thus, reasonably-performing models can reduce the search space considerably, making a brute-force approach to secret-key discovery feasible.

From looking at the confusion matrices, we detect slight biases of the 8×16 hashing matrix towards a certain resulting bit in some of its rows, which can lead to slight performance biases, as mentioned in section 4.2.1. Nevertheless, by inspecting the confusion matrices for the scenarios of each hashing-matrix size, we conclude these should be practically negligible.

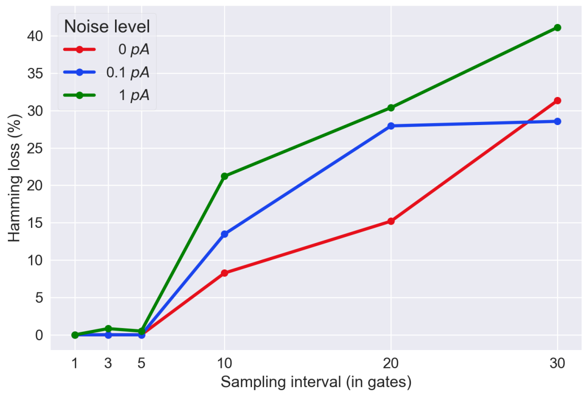
Let us now summarize the average hamming loss of the best full-matrix models for each scenario, by sampling interval, in figures 4.11a to 4.11c.



(a) For a 8×16 hashing matrix.



(b) For a 16×32 hashing matrix.



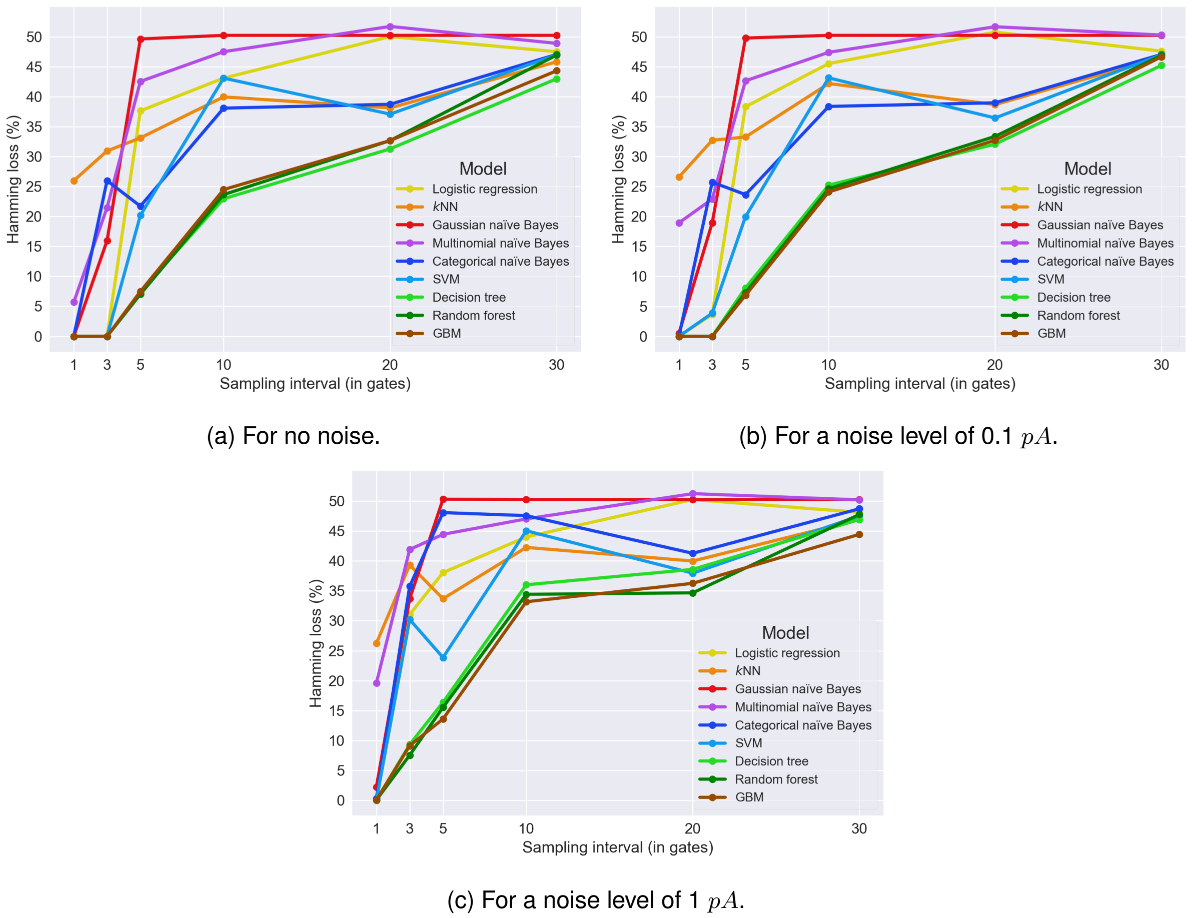
(c) For a 32×64 hashing matrix.

Figure 4.11: Average Hamming loss of the predicted keys of the best full-matrix model by sampling interval, for each noise level and hashing-matrix size.

There was no significant degradation with the increase of the hashing-matrix size for sampling intervals up to 5 gates. In these scenarios, the degradation with the increase in noise level was only significant for a sampling interval of 3 gates. Nevertheless, in those cases, having a noise level of 1 pA only slightly increased the Hamming loss by around 1%. The degradation for sampling intervals above 5 gates was much higher.

4.2.3 Row models

Completing the same steps as in the full-matrix case, but for the row models, we now present the average Hamming loss of their predicted keys, by sampling interval and noise level, for a 32×64 hashing matrix, in figures 4.12a to 4.12c. The remaining scenarios are presented in appendix B.3, which we also consider in the analysis below.



(a) For no noise. (b) For a noise level of 0.1 pA.

(c) For a noise level of 1 pA.

Figure 4.12: Average Hamming loss of the predicted keys of row models by sampling interval, for each noise level, with a 32×64 hashing matrix.

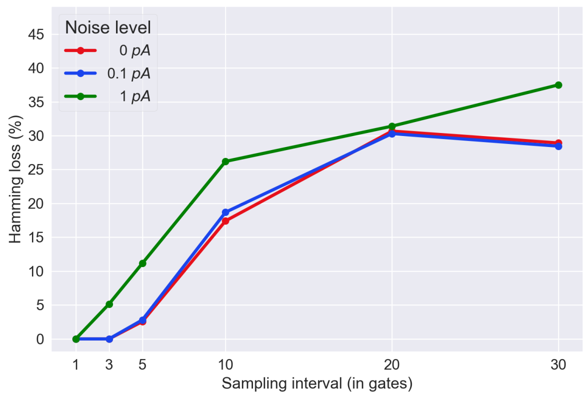
In most scenarios, the best performance was divided between GBM, random forest and decision tree. While the results are generally not as satisfactory as in the full-matrix case, most models were able to predict the full secret key when sampling all gates, and a few still succeeded when sampling 1 of every 3 gates for noise levels up to 0.1 pA. However, for instance, when we increased the noise level from 0.1 pA to 1 pA, with a 32×64 hashing matrix and a sampling interval of 3 gates, the Hamming loss of the best model, the random forest, increased from 0% to 7.54%. For sampling intervals above 5 gates, the performance dropped substantially.

There was a large discrepancy in performance between the scenarios where we sampled all gates and the ones we did not. For instance, in the scenarios above, the Gaussian naïve Bayes went from correctly predicting the full secret key almost 100% of the time, when sampling all gates, to being as useful as a coin flip for deciding each bit of the predicted key, for sampling intervals of 5 gates or more.

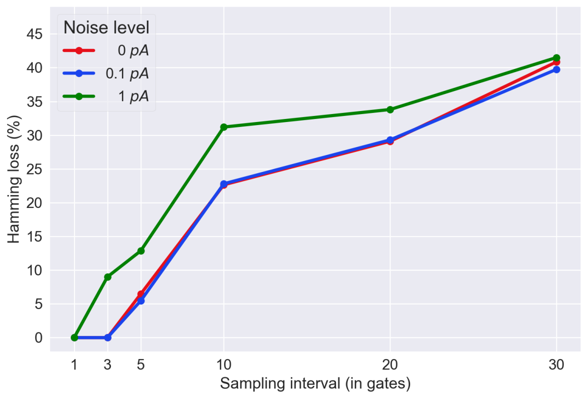
Based on these results, it seems that, despite the reduction in irrelevant data fed to the models, the introduction of the neutral data point in the subrows with one less data point, which happens for sampling intervals larger than 1 gate, worsens the performance of the models considerably.

The worst-performing models across all scenarios with sampling intervals larger than 1 gate were the Gaussian and multinomial naïve Bayes and logistic regression. We note that kNN and the multinomial naïve Bayes were the only models not able to consistently predict the full secret key in any scenario, not even when sampling all gates.

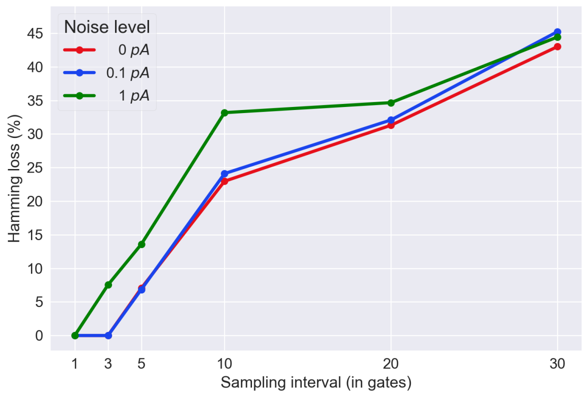
Let us examine the average Hamming loss of the predicted keys of the best row model for each scenario, plotted by sampling interval and shown in figures 4.13a to 4.13c.



(a) For a 8×16 hashing matrix.



(b) For a 16×32 hashing matrix.



(c) For a 32×64 hashing matrix.

Figure 4.13: Average Hamming loss of the predicted keys of the best row model by sampling interval, for each noise level and hashing-matrix size.

There was no performance degradation with the increase of the hashing-matrix size or noise level when sampling all gates. The same was true for a sampling interval of 3 gates, when the noise level did not go above 0.1 pA. In the remaining scenarios, the degradation with the increase of the hashing-matrix size was not very substantial, while the degradation with the noise level was only significant for high noise levels of 1 pA.

The problem of the sampled-gate subset is apparent for the 8×16 hashing matrix and low noise scenarios, with the best models for a sampling interval of 20 gates having a worse performance than the models for a sampling interval of 30 gates.

4.3 Attack-approach comparison

We shall now compare the machine-learning approaches tested.

Firstly, we note that the full-matrix models were generally more successful at predicting the full secret key than the 1-bit models were at extracting only 1 bit from the power trace. Moreover, the degradation with the increase of the hashing-matrix size or noise level also appears to be less severe for the full-matrix models than in the 1-bit test case. Nevertheless, we must account for the fact that the full-matrix models are, in fact, ensembles of l models, each specialized in predicting one of the l bits of the secret key, while the 1-bit models are single models which must extract information equivalent to knowing there are at least $\lfloor \frac{l}{2} + 1 \rfloor$ ones or zeros in the secret key to confidently predict which bit is more common. Therefore, while a 1-bit model needs less information than the full-matrix model ensemble, it requires more information than each individual model of that ensemble, which may explain the worse results of the test case.

GBM performed generally well in all approaches, with other models being marginally better in a few scenarios. Thus, if there is a lack of time to perform a similar thorough analysis of a specific physical implementation to know which model to use in a certain scenario, Eve would only use a GBM.

The full-matrix approach had generally better results than the row approach, with the row models having more difficulty in distinguishing and classifying the power trace of a single logical scalar product than the full-matrix models in classifying the full power trace. Furthermore, the full-matrix models were able to fully predict the secret key with near certainty for sampling intervals of up to 5 gates while the row models could only do it for sampling intervals up to 3 gates and only if the noise level was low. However, the full-matrix approach is significantly more computationally expensive since the models are trained with much more data. For instance, some of the full-matrix models took a full day to be trained on a computer with a 4-core CPU and 8 Gb of RAM, while the row models took at most around 5 hours. Therefore, the choice of approach usually implies a trade-off between the computational resources needed and the attack performance.

Nevertheless, most of the row models were able to predict the secret key with virtual certainty when sampling all gates, while only GBM, SVM and logistic regression were able to do it in the same scenarios with the full-matrix approach. Thus, Eve could use the row approach if she is sure the measuring instrument is able to sample all gates, since this approach works equally well or even better than the full-matrix approach due to using less irrelevant data, and is less time consuming since the models use much less features. If she was not able to sample all gates, then she would use the full-matrix approach, provided she had enough computational resources.

We would like to note that we also tested the full-matrix approach with GBM for a privacy-amplification protocol with a 32×64 randomly-generated binary hashing matrix with no entry restrictions, noise level of 0.1 pA and the same sampling intervals as the previous scenarios, and the results were similar to the 32×64 modified Toeplitz matrix scenarios with the same noise level. Thus, it is highly likely that this attack can be successful for general binary hashing matrices.

4.4 Countermeasures

The results of this analysis make it clear that a similar side-channel attack to the one we proposed can often be successful, which renders any security proof obsolete, being equivalent to $\epsilon_{sec} \to 1$ in the secrecy condition (2.21). Therefore, it is necessary to consider countermeasures for similar attacks.

As mentioned in section 3.2, countermeasures can either be applied on the devices of an implementation or on the protocol itself. The first case usually focuses on reducing the side-channel leakage, while the second focuses on eliminating the correlation between the leakage and the secret key.

Noise insertion As mentioned before, a high-enough noise level can always prevent the success of a side-channel attack similar to ours. Thus, a possible countermeasure to such an attack is to attach noise sources to the vulnerable devices of an implementation and make them generate noise in parallel to the main computations. However, it must be ensured that the noise level is high enough. Otherwise, statistical techniques, such as DPA, can be used to isolate the signal from the noise. Moreover, a skilled-enough Eve may be able to circumvent the noise sources if she has access to the implementation. Therefore, proper shielding should also be used to prevent access to the internal circuitry of the implementation devices.

Another option is to modify the protocol to perform randomly-timed dummy operations during the main computations. This acts as noise and successfully hides the main computations from Eve without being easily circumvented. However, it may also lead to a significant reduction in key rate.

Masking Masking is also known as secret sharing [166] in a more general context. It consists in hiding secret data by not processing it directly. More specifically, Alice and Bob hide the secret key by processing only a share of it at a time. In the context of privacy amplification with matrix hashing, this consists in splitting the corrected key into d components as

$$\boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{x}_2 + ... + \boldsymbol{x}_d \mod 2, \tag{4.1}$$

where each x_i is called a share of the corrected key, and then multiplying the hashing matrix with each share. The secret key can be reconstructed by logical addition of all the shares due to linearity of the operations.

The most common use of this technique is called first-order masking [167] and consists in splitting the key into two shares, known as the mask and the masked key.

It was shown in [166] that the share division can be made such that complete reconstruction of $d - 1$ shares reveals no information about the secret key. Thus, Eve would have to measure leakages from the processing of all shares to be able to reconstruct the secret key.

Moreover, [167] showed that masking can be combined with noise to provide information-theoretical security. They proved that, for a certain amount of noise in the leakages, the secret-key information which Eve can obtain from all leakages of one protocol execution is upper bounded by a value which decreases exponentially with the increase in the number of key shares.

As with simple noise insertion, proper shielding can be crucial for Eve not to circumvent the noise sources.

Randomization Randomization techniques consist in adding additional steps to the protocol which randomly shuffle the order of operations to eliminate the correlation between side channels and the secret key.

A possible randomization countermeasure for privacy-amplification protocols based on matrix hashing is for Alice and Bob to apply independent secret random permutations on the rows of the hashing matrix before its logical multiplication with the corrected key. This shuffles the order of the logical scalar products corresponding to each row of the hashing matrix, which renders the full-matrix approach obsolete. However, if Eve uses the row approach, she can still know the number of zeros and ones of the secret key, l_0 and l_1, respectively. Then, her search space will be reduced from 2^l possibilities to $\binom{l}{l_0}$. Since this attack targets the privacy-amplification step, no further key distillation can be performed to remove Eve's knowledge about the secret key.

Another randomization technique is for Alice and Bob to apply independent secret random permutations on the columns of the hashing matrix and on the corrected key before computing their logical multiplication. This effectively renders both approaches obsolete. All Eve can do is use brute force and build a different machine-learning model for every possible permutation of either Alice or Bob. For a corrected key with n bits, assuming all columns of the hashing matrix are different, she will have to build up to $n!$ models. This is usually a much larger quantity than the number of possibilities for the secret key, 2^l. Thus, this countermeasure recovers the information-theoretical security of the QKD protocol, even when considering attacks similar to ours, and should be preferred over the previous countermeasure. However, it is possible Eve can circumvent it by measuring the power trace of the permutation operations to gain knowledge on the permutation applied.

Even more aggressive measures are for Alice and Bob to independently apply a different random permutation on the columns of the hashing matrix and on the corrected key for each row of the hashing matrix, or for them to independently apply random permutations to both columns and rows of the hashing matrix. However, these would not increase the security further and require them to generate and apply multiple truly-random permutations, which can decrease the key rate unnecessarily.

Yet another countermeasure is to process multiple logical scalar products in parallel to scramble side-channel leakages without necessarily decreasing the key rate.

Properly implementing these countermeasures might not be enough to deter potential eavesdroppers from finding new side-channel attacks. Thus, constant surveillance of both the communicating parties' setups and the multiple repeater locations in a QKD network is likely advisable to prevent any type of unwanted access to the physical implementations.

Importantly, we note that, if security is the main priority, we should always rely on a countermeasure which may potentially lead to a decrease in key rate rather than on a "large-enough" secret key, since a powerful-enough computer may always be able to break our protocol in the latter case. Besides, the advantage of using a QKD protocol is having information-theoretical security. By relying on a large secret key without implementing adequate countermeasures, we may be again reduced to computational security, as in public-key cryptosystems.

Chapter 5

Conclusions

In this chapter, we conclude this thesis by describing our main achievements in section 5.1 and discussing further considerations and possible future work in sections 5.2 and 5.3, respectively.

5.1 Achievements

In this thesis, we aimed to contribute to the research on classical-side-channel attacks on QKD implementations, while also reviewing some of the main theoretical and experimental progress in the field of QKD.

Our main achievements were:

1. a brief review of some of the major theoretical and experimental advances in QKD, namely:
   (a) a review of the main breakthroughs in security-analysis techniques;
   (b) a description of the main components of QKD implementations, the most common DV-QKD implementations and the main algorithms for classical postprocessing;
   (c) a description of the use of classical and quantum repeaters for extending the transmission distance in QKD;
   (d) a comprehensive review of the main quantum- and classical-side-channel attacks demonstrated in QKD implementations and their respective countermeasures;
2. an introduction to the field of machine learning in a mostly-self-contained way, focusing on supervised learning, to provide all the necessary background to understand our main contribution;
3. a new classical-side-channel attack on the privacy-amplification step of a general QKD protocol based on matrix hashing, using its power-consumption trace;

   Moreover, we presented frameworks for simulating the power trace of a protocol based on matrix multiplication and for analysing a leakage with machine-learning techniques. We applied the latter to the simulated power traces.

We simulated our attack in multiple scenarios with different sizes of the modified Toeplitz hashing matrix, measurement noise levels and sampling intervals of the measuring instrument, which is a time-independent way of describing the instrument sampling rate. For all scenarios, we tested different variations of the machine-learning analysis to assess the best approach for the eavesdropper, Eve. We started with a test case in which we assessed the ability of the machine-learning models to extract 1 bit of information from the full power trace. Then, we tried to recover the full secret key by providing the machine-learning models with power traces corresponding to operations involving either the full hashing matrix or just the row of the hashing matrix used in the computation of the secret-key bit we were trying to predict. We then analysed the quality of the generated power traces and the results of our three approaches to the machine-learning analysis for each protocol scenario.

GBM was the best-performing model in virtually every scenario. It was able to recover the full secret key for small-enough sampling intervals, equivalent to high-enough sampling rates, with any hashing-matrix size and noise level tested.

Models provided with power traces corresponding to the full hashing matrix performed generally better than models provided with power traces corresponding to individual rows of the hashing matrix. However, the former models require more computational resources than the latter which leads to a trade-off between the computational resources needed and the attack performance.

Within our simulated scenarios, there was no significant degradation of the performance of the best model with the increase of either the hashing-matrix size or the noise level, provided the sampling interval was small enough. However, we note that a high-enough noise level should always be able to hinder an attack similar to ours. Moreover, it is possible that the performance of a machine-learning model degrades for a large-enough hashing matrix.

We also determined that the sampling interval plays a major role in the performance of the models, with larger sampling intervals sometimes leading to better-performing models due to the particular sample of the power trace acquired by Eve.

Moreover, we devised a strategy, based on analysing the confusion matrices of the model, which can make a constrained brute-force search for the secret key feasible in case of non-perfect models.

Finally, we also tested our analysis with a general binary hashing matrix and the results were similar to the scenarios with a modified Toeplitz hashing matrix.

This demonstrates that machine-learning techniques can be used to robustly characterize the leakages in a QKD implementation and generate powerful attacks.

4. Multiple countermeasures to attacks similar to the one we proposed, based on noise insertion, masking and randomization techniques.

   Some of the proposed countermeasures are capable of restoring the information-theoretical security of the QKD protocol when considering attacks similar to ours, namely masking the key in combination with noise, and applying a secret random permutation on the columns of the hashing matrix and on the corrected key before computing their logical multiplication.

Moreover, we note that the proposed countermeasures based on masking and randomization techniques should also be successful at eliminating other potential side channels in classical-postprocessing devices, such as emanated electromagnetic radiation.

This work also led to the paper [168], currently awaiting publishing.

5.2 Further considerations

A quantum cryptosystem whose security relies solely on quantum postulates is currently unrealistic. Most physical implementations are imperfect and naturally present both classical and quantum side channels. Our work reinforces that classical side channels in implementations must be taken into account for practical use of QKD. This is specially true for QKD networks with multiple remotely-located repeaters, since these can be particularly exposed to attacks.

We note that this argument remains relevant for MDI- and DI-QKD implementations. MDI-QKD protocols treat the untrusted measurement relay as a true black box, such that any information leakage from it cannot reveal significant information about the secret key. Thus, these protocols are also free from classical side channels in the detection components. However, the source components used by Alice and Bob can be vulnerable to both classical and quantum side channels. DI-QKD arguably eliminates all quantum side channels from its implementations through device self-testing. However, their components can still be vulnerable to classical side channels, since these do not destroy quantum correlations in the shared systems.

We note that there may be a significant amount of exploitable side channels which have not yet been discovered. Therefore, besides the numerous challenges and open questions regarding QKD currently being explored [5], greater focus should be given to research on potential side-channel attacks if we hope to consider QKD as a truly-secure technology in the future.

Nevertheless, we emphasize that QKD still has a clear advantage over classical systems based on computational security, since the security of the former only depends on Eve successfully performing a side-channel attack, while the latter can be broken either by side-channel attacks or by Eve saving the encrypted messages until she has enough computational resources to decrypt them.

5.3 Future work

The attack in this work was performed on simulated power traces due to lack of access to physical hardware. A natural next step is to use the same machine-learning framework to analyse power-consumption traces from a physical implementation to verify the validity of our analysis. We believe that this can only strengthen our argument for considering classical side channels in QKD implementations.

Moreover, a more in-depth study of the effects of the hashing-matrix size and noise level in similar attacks would be useful to understand the strength of such attacks. More precisely, one could investigate if there is any performance degradation for the matrix sizes used in practical implementations and determine the noise-level threshold above which it is no longer possible to extract useful information from similar power traces.

It would also be useful to analyse the amount of information Eve can obtain from measuring the power consumption of the permutation operations in the randomization countermeasure proposed to verify if such countermeasure is easily circumvented.

Besides power consumption, our attack framework can be used to exploit other side channels, such as electromagnetic radiation emanated from implementation devices and even their cache state throughout the QKD protocol. We believe all potential side channels should be thoroughly explored.

Finally, it would be useful to perform a study on the strength of machine-learning-based side-channel attacks and how they compare to template attacks, which are assumed to be the strongest type of side-channel attack and are thus commonly used to evaluate the security limits of cryptographic implementations.

Bibliography

[1] H. Delfs and H. Knebl. *Introduction to Cryptography*. 3rd. Information Security and Cryptography. Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-47973-5. DOI: 10.1007/978-3-662-47974-2.

[2] W. Diffie and M. Hellman. “New directions in cryptography”. *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638.

[3] R. L. Rivest et al. “A method for obtaining digital signatures and public-key cryptosystems”. *Communications of the ACM* 21.2 (1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342.

[4] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. 10th. Cambridge University Press, 2010. ISBN: 9781107002173. DOI: 10.1017/CBO9780511976667.

[5] S. Pirandola et al. “Advances in quantum cryptography”. *Advances in Optics and Photonics* 12.4 (2020), p. 1012. ISSN: 19438206. DOI: 10.1364/aop.361502.

[6] H.-K. Lo and Y. Zhao. “Quantum Cryptography”. *Encyclopedia of Complexity and Systems Science*. Springer New York, 2009, pp. 7265–7289. DOI: 10.1007/978-0-387-30440-3_432.

[7] J.-P. Chen et al. “Sending-or-Not-Sending with Independent Lasers: Secure Twin-Field Quantum Key Distribution over 509 km”. *Physical Review Letters* 124.7 (2020), p. 070501. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.124.070501.

[8] L. Salvail et al. *Security of Trusted Repeater Quantum Key Distribution Networks*. 2009. arXiv: 0904.4072 [quant-ph].

[9] H.-J. Briegel et al. “Quantum Repeaters: The Role of Imperfect Local Operations in Quantum Communication”. *Physical Review Letters* 81.26 (1998), pp. 5932–5935. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.81.5932.

[10] P. Kocher et al. “Differential Power Analysis”. *Advances in Cryptology — CRYPTO’ 99*. Springer Berlin Heidelberg, 1999, pp. 388–397. ISBN: 978-3-540-48405-9. DOI: 10.1007/3-540-48405-1_25.

[11] K. Gandolfi et al. “Electromagnetic Analysis: Concrete Results”. *Cryptographic Hardware and Embedded Systems — CHES 2001*. Springer Berlin Heidelberg, 2001, pp. 251–261. ISBN: 978-3-540-44709-2. DOI: 10.1007/3-540-44709-1_21.

[12] D. Genkin et al. “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”. *Advances in Cryptology – CRYPTO 2014*. Springer Berlin Heidelberg, 2014, pp. 444–461. ISBN: 978-3-662-44371-2. DOI: 10.1007/978-3-662-44371-2_25.

[13] H.-K. Lo et al. “Measurement-Device-Independent Quantum Key Distribution”. *Physical Review Letters* 108.13 (2012), p. 130503. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.108.130503.

[14] A. Acín et al. “Device-Independent Security of Quantum Cryptography against Collective Attacks”. *Physical Review Letters* 98.23 (2007), p. 230501. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.98.230501.

[15] A. Lamas-Linares and C. Kurtsiefer. “Breaking a quantum key distribution system through a timing side channel”. *Optics Express* 15.15 (2007), p. 9388. ISSN: 10944087. DOI: 10.1364/oe.15.009388.

[16] D. Park et al. “Single trace side-channel attack on key reconciliation in quantum key distribution system and its efficient countermeasures”. *ICT Express* 7.1 (2021), pp. 36–40. ISSN: 24059595. DOI: 10.1016/j.icte.2021.01.013.

[17] G. Kim et al. “Side Channel Vulnerability in Parity Computation of Generic Key Reconciliation Process on QKD”. *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2021, pp. 257–261. ISBN: 978-1-6654-2383-0. DOI: 10.1109/ICTC52510.2021.9620820.

[18] S. Kim et al. “Single Trace Side Channel Analysis on Quantum Key Distribution”. *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2018, pp. 736–739. ISBN: 978-1-5386-5041-7. DOI: 10.1109/ICTC.2018.8539703.

[19] K. Durak and N. Jam. *An attack to quantum systems through RF radiation tracking*. 2020. arXiv: 2004.14445 [quant-ph].

[20] A. Weber et al. “Cache-Side-Channel Quantification and Mitigation for Quantum Cryptography”. *Computer Security – ESORICS 2021*. Springer International Publishing, 2021, pp. 235–256. ISBN: 978-3-030-88427-7. DOI: 10.1007/978-3-030-88428-4_12.

[21] M. M. Wilde. *Quantum Information Theory*. 2nd. Cambridge University Press, 2017. ISBN: 9781316809976. DOI: 10.1017/9781316809976.

[22] M. Tomamichel and A. Leverrier. “A largely self-contained and complete security proof for quantum key distribution”. *Quantum* 1 (2017), p. 14. ISSN: 2521-327X. DOI: 10.22331/q-2017-07-14-14.

[23] J. Carter and M. N. Wegman. “Universal classes of hash functions”. *Journal of Computer and System Sciences* 18.2 (1979), pp. 143–154. ISSN: 00220000. DOI: 10.1016/0022-0000(79)90044-8.

[24] D. R. Stinson. “Universal hashing and authentication codes”. *Designs, Codes and Cryptography* 4.3 (1994), pp. 369–380. ISSN: 0925-1022. DOI: 10.1007/BF01388651.

[25] R. Renner and S. Wolf. “Unconditional Authenticity and Privacy from an Arbitrarily Weak Secret”. *Advances in Cryptology - CRYPTO 2003*. Springer Berlin Heidelberg, 2003, pp. 78–95. ISBN: 978-3-540-45146-4. DOI: 10.1007/978-3-540-45146-4_5.

[26] M. N. Wegman and J. Carter. “New hash functions and their use in authentication and set equality”. *Journal of Computer and System Sciences* 22.3 (1981), pp. 265–279. ISSN: 00220000. DOI: 10.1016/0022-0000(81)90033-7.

[27] C. H. Bennett and G. Brassard. “Quantum cryptography: Public key distribution and coin tossing”. *Theoretical Computer Science* 560 (2014), pp. 7–11. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2014.05.025.

[28] C. H. Bennett et al. “Quantum cryptography without Bell’s theorem”. *Physical Review Letters* 68.5 (1992), pp. 557–559. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.68.557.

[29] A. K. Ekert. “Quantum cryptography based on Bell’s theorem”. *Physical Review Letters* 67.6 (1991), pp. 661–663. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.67.661.

[30] J. F. Clauser et al. “Proposed Experiment to Test Local Hidden-Variable Theories”. *Physical Review Letters* 23.15 (1969), pp. 880–884. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.23.880.

[31] C. H. Bennett. “Quantum cryptography using any two nonorthogonal states”. *Physical Review Letters* 68.21 (1992), pp. 3121–3124. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.68.3121.

[32] H. Bechmann-Pasquinucci and N. Gisin. “Incoherent and coherent eavesdropping in the six-state protocol of quantum cryptography”. *Physical Review A* 59.6 (1999), pp. 4238–4248. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.59.4238.

[33] V. Scarani et al. “Quantum Cryptography Protocols Robust against Photon Number Splitting Attacks for Weak Laser Pulse Implementations”. *Physical Review Letters* 92.5 (2004), p. 057901. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.92.057901.

[34] M. Tomamichel and E. Hänggi. “The link between entropic uncertainty and nonlocality”. *Journal of Physics A: Mathematical and Theoretical* 46.5 (2013), p. 055301. ISSN: 1751-8113. DOI: 10.1088/1751-8113/46/5/055301.

[35] H.-K. Lo and H. F. Chau. “Unconditional Security of Quantum Key Distribution over Arbitrarily Long Distances”. *Science* 283.5410 (1999), pp. 2050–2056. ISSN: 0036-8075. DOI: 10.1126/science.283.5410.2050.

[36] P. W. Shor and J. Preskill. “Simple Proof of Security of the BB84 Quantum Key Distribution Protocol”. *Physical Review Letters* 85.2 (2000), pp. 441–444. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.85.441.

[37] D. Mayers. “Unconditional security in quantum cryptography”. *Journal of the ACM* 48.3 (2001), pp. 351–406. ISSN: 0004-5411. DOI: 10.1145/382780.382781.

[38] D. Mayers. “Quantum Key Distribution and String Oblivious Transfer in Noisy Channels”. *Advances in Cryptology — CRYPTO ’96*. Springer Berlin Heidelberg, 1996, pp. 343–357. DOI: 10.1007/3-540-68697-5_26.

[39] N. Lütkenhaus. “Security against individual attacks for realistic quantum key distribution”. *Physical Review A* 61.5 (2000), p. 052304. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.61.052304.

[40] H. Inamori et al. “Unconditional security of practical quantum key distribution”. *The European Physical Journal D* 41.3 (2007), pp. 599–627. ISSN: 1434-6060. DOI: 10.1140/epjd/e2007-00010-4.

[41] B. Huttner et al. “Quantum cryptography with coherent states”. *Physical Review A* 51.3 (1995), pp. 1863–1869. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.51.1863.

[42] D. Gottesman et al. “Security of quantum key distribution with imperfect devices”. *Quantum Information and Computation* 4.5 (2004), pp. 325–360. ISSN: 15337146. DOI: 10.26421/QIC4.5-1.

[43] M. Koashi. “Unconditional security of quantum key distribution and the uncertainty principle”. *Journal of Physics: Conference Series* 36 (2006), pp. 98–102. ISSN: 1742-6588. DOI: 10.1088/1742-6596/36/1/016.

[44] H. Maassen and J. B. M. Uffink. “Generalized entropic uncertainty relations”. *Physical Review Letters* 60.12 (1988), pp. 1103–1106. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.60.1103.

[45] T. Tsurumaru. “Leftover Hashing From Quantum Error Correction: Unifying the Two Approaches to the Security Proof of Quantum Key Distribution”. *IEEE Transactions on Information Theory* 66.6 (2020), pp. 3465–3484. ISSN: 0018-9448. DOI: 10.1109/TIT.2020.2969656.

[46] M. Koashi. “Simple security proof of quantum key distribution based on complementarity”. *New Journal of Physics* 11.4 (2009), p. 045018. ISSN: 1367-2630. DOI: 10.1088/1367-2630/11/4/045018.

[47] M. Ben-Or et al. “The Universal Composable Security of Quantum Key Distribution”. *Theory of Cryptography*. Springer Berlin Heidelberg, 2005, pp. 386–406. ISBN: 978-3-540-30576-7. DOI: 10.1007/978-3-540-30576-7_21.

[48] R. Renner and R. König. “Universally Composable Privacy Amplification Against Quantum Adversaries”. *Theory of Cryptography*. Springer Berlin Heidelberg, 2005, pp. 407–425. ISBN: 978-3-540-30576-7. DOI: 10.1007/978-3-540-30576-7_22.

[49] R. Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Paper 2000/067. 2000. URL: https://eprint.iacr.org/2000/067.

[50] R. Renner. “Security of Quantum Key Distribution”. Doctoral thesis. *International Journal of Quantum Information* 06.01 (2008), pp. 1–127. ISSN: 0219-7499. DOI: 10.1142/S0219749908003256.

[51] M. Tomamichel et al. “Tight finite-key analysis for quantum cryptography”. *Nature Communications* 3.1 (2012), p. 634. ISSN: 2041-1723. DOI: 10.1038/ncomms1631.

[52] M. Hayashi and T. Tsurumaru. “Concise and tight security analysis of the Bennett–Brassard 1984 protocol with finite key lengths”. *New Journal of Physics* 14.9 (2012), p. 093014. ISSN: 1367-2630. DOI: 10.1088/1367-2630/14/9/093014.

[53] M. Tomamichel and R. Renner. “Uncertainty Relation for Smooth Entropies”. *Physical Review Letters* 106.11 (2011), p. 110506. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.106.110506.

[54] C. Pfister et al. “Sifting attacks in finite-size quantum key distribution”. *New Journal of Physics* 18.5 (2016), p. 053001. ISSN: 1367-2630. DOI: 10.1088/1367-2630/18/5/053001.

[55] F. Dupuis et al. “Entropy Accumulation”. *Communications in Mathematical Physics* 379.3 (2020), pp. 867–913. ISSN: 0010-3616. DOI: 10.1007/s00220-020-03839-5.

[56] R. Arnon-Friedman et al. “Simple and Tight Device-Independent Security Proofs”. *SIAM Journal on Computing* 48.1 (2019), pp. 181–225. ISSN: 0097-5397. DOI: 10.1137/18M1174726.

[57] F. Dupuis and O. Fawzi. “Entropy Accumulation With Improved Second-Order Term”. *IEEE Transactions on Information Theory* 65.11 (2019), pp. 7596–7612. ISSN: 0018-9448. DOI: 10.1109/TIT.2019.2929564.

[58] F. Xu et al. “Secure quantum key distribution with realistic devices”. *Reviews of Modern Physics* 92.2 (2020), p. 025002. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.92.025002.

[59] N. Gisin et al. “Quantum cryptography”. *Reviews of Modern Physics* 74.1 (2002), pp. 145–195. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.74.145.

[60] M. Leifgen. “Protocols and components for quantum key distribution”. Doctoral thesis. Humboldt-Universität zu Berlin, 2016. DOI: http://dx.doi.org/10.18452/17473.

[61] L. Mach. “Ueber einen interferenzrefraktor”. (in German). *Zeitschrift für Instrumentenkunde* 12.3 (1892), pp. 89–93.

[62] S.-K. Liao et al. “Satellite-to-ground quantum key distribution”. *Nature* 549.7670 (2017), pp. 43–47. ISSN: 0028-0836. DOI: 10.1038/nature23655.

[63] B. E. Kardynał et al. “An avalanche-photodiode-based photon-number-resolving detector”. *Nature Photonics* 2.7 (2008), pp. 425–428. ISSN: 1749-4885. DOI: 10.1038/nphoton.2008.101.

[64] X.-F. Mo et al. “Faraday–Michelson system for quantum cryptography”. *Optics Letters* 30.19 (2005), p. 2632. ISSN: 0146-9592. DOI: 10.1364/OL.30.002632.

[65] H.-Q. Ma et al. “Quantum key distribution based on phase encoding and polarization measurement”. *Optics Letters* 32.6 (2007), p. 698. ISSN: 0146-9592. DOI: 10.1364/OL.32.000698.

[66] A. Muller et al. ““Plug and play” systems for quantum cryptography”. *Applied Physics Letters* 70.7 (1997), pp. 793–795. ISSN: 0003-6951. DOI: 10.1063/1.118224.

[67] G. Brassard and L. Salvail. “Secret-Key Reconciliation by Public Discussion”. *Advances in Cryptology — EUROCRYPT ’93*. Vol. 765. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1994, pp. 410–423. DOI: 10.1007/3-540-48285-7_35.

[68] C. H. Bennett et al. “Experimental quantum cryptography”. *Journal of Cryptology* 5.1 (1992), pp. 3–28. ISSN: 0933-2790. DOI: 10.1007/BF00191318.

[69] L. Hu et al. “Parameter optimization of cascade in quantum key distribution”. *Optik* 181 (2019), pp. 156–162. ISSN: 00304026. DOI: 10.1016/j.ijleo.2018.12.023.

[70] E. Bai et al. “Cascade protocol with parameters and backtracking optimization”. *IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*. IEEE, 2021, pp. 571–574. ISBN: 978-1-7281-9004-4. DOI: 10.1109/ICPECA51329.2021.9362598.

[71] T. B. Pedersen and M. Toyran. *High Performance Information Reconciliation for QKD with CASCADE*. 2013. arXiv: 1307.7829 [quant-ph].

[72] A. Reis. “Quantum Key Distribution Post Processing - A study on the Information Reconciliation Cascade Protocol”. Master’s thesis. University of Porto, 2019. URL: https://hdl.handle.net/10216/121965.

[73] H.-K. Mao et al. “High performance reconciliation for practical quantum key distribution systems” (2021). arXiv: 2101.12565 [quant-ph].

[74] W. T. Buttler et al. “Fast, efficient error reconciliation for quantum cryptography”. *Physical Review A* 67.5 (2003), p. 052303. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.67.052303.

[75] R. G. Gallager. “Low Density Parity Check Codes”. Doctoral thesis. Massachusetts Institute of Technology, 1960. URL: http://hdl.handle.net/1721.1/11804.

[76] R. Tanner. “A recursive approach to low complexity codes”. *IEEE Transactions on Information Theory* 27.5 (1981), pp. 533–547. ISSN: 0018-9448. DOI: 10.1109/TIT.1981.1056404.

[77] D. Elkouss et al. “Information reconciliation for QKD”. *Quantum Information and Computation* 11.3&4 (2011), pp. 226–238. ISSN: 15337146. DOI: 10.26421/QIC11.3-4-3.

[78] Xiao-Yu Hu et al. “Regular and irregular progressive edge-growth tanner graphs”. *IEEE Transactions on Information Theory* 51.1 (2005), pp. 386–398. ISSN: 0018-9448. DOI: 10.1109/TIT.2004.839541.

[79] J. Martinez-Mateo et al. “Improved Construction of Irregular Progressive Edge-Growth Tanner Graphs”. *IEEE Communications Letters* 14.12 (2010), pp. 1155–1157. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2010.101810.101384.

[80] M. Fossorier et al. “Reduced complexity iterative decoding of low-density parity check codes based on belief propagation”. *IEEE Transactions on Communications* 47.5 (1999), pp. 673–680. ISSN: 00906778. DOI: 10.1109/26.768759.

[81] D. Elkouss et al. “Efficient reconciliation protocol for discrete-variable quantum key distribution”. *IEEE International Symposium on Information Theory*. IEEE, 2009, pp. 1879–1883. ISBN: 978-1-4244-4312-3. DOI: 10.1109/ISIT.2009.5205475.

[82] J. Martinez-Mateo et al. “Blind reconciliation”. *Quantum Information and Computation* 12.9&10 (2012), pp. 791–812. ISSN: 15337146. DOI: 10.26421/QIC12.9-10-5.

[83] E. O. Kiktenko et al. “Symmetric Blind Information Reconciliation for Quantum Key Distribution”. *Physical Review Applied* 8.4 (2017), p. 044017. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.8.044017.

[84] C. Gao et al. “Multi-matrix error estimation and reconciliation for quantum key distribution”. *Optics Express* 27.10 (2019), p. 14545. ISSN: 1094-4087. DOI: 10.1364/OE.27.014545.

[85] K. Nguyen et al. “Side-Information Coding with Turbo Codes and its Application to Quantum Key Distribution”. *International Symposium on Information Theory and its Applications*. 2004. arXiv: cs/0406001 [cs.IT].

[86] P. Jouguet and S. Kunz-Jacques. “High performance error correction for quantum key distribution using polar codes”. *Quantum Information and Computation* 14.3&4 (2014), pp. 329–338. ISSN: 15337146. DOI: 10.26421/QIC14.3-4-8.

[87] M. Mehic et al. “Error Reconciliation in Quantum Key Distribution Protocols”. *Reversible Computation: Extending Horizons of Computing*. Springer International Publishing, 2020, pp. 222–236. ISBN: 978-3-030-47361-7. DOI: 10.1007/978-3-030-47361-7_11.

[88] C. H. Bennett et al. “Privacy Amplification by Public Discussion”. *SIAM Journal on Computing* 17.2 (1988), pp. 210–229. ISSN: 0097-5397. DOI: 10.1137/0217014.

[89] C. Bennett et al. “Generalized privacy amplification”. *IEEE Transactions on Information Theory* 41.6 (1995), pp. 1915–1923. ISSN: 00189448. DOI: 10.1109/18.476316.

[90] M. Tomamichel et al. “Leftover Hashing Against Quantum Side Information”. *IEEE Transactions on Information Theory* 57.8 (2011), pp. 5524–5535. ISSN: 0018-9448. DOI: 10.1109/TIT.2011.2158473.

[91] H. Krawczyk. “LFSR-based Hashing and Authentication”. *Advances in Cryptology — CRYPTO ’94*. Springer Berlin Heidelberg, 1994, pp. 129–139. ISBN: 978-3-540-48658-9. DOI: 10.1007/3-540-48658-5_15.

[92] M. Hayashi. “Exponential Decreasing Rate of Leaked Information in Universal Random Privacy Amplification”. *IEEE Transactions on Information Theory* 57.6 (2011), pp. 3989–4001. ISSN: 0018-9448. DOI: 10.1109/TIT.2011.2110950.

[93] W. K. Wootters and W. H. Zurek. “A single quantum cannot be cloned”. *Nature* 299.5886 (1982), pp. 802–803. ISSN: 0028-0836. DOI: 10.1038/299802a0.

[94] S. Pirandola et al. “High-rate measurement-device-independent quantum cryptography”. *Nature Photonics* 9.6 (2015), pp. 397–402. ISSN: 1749-4885. DOI: 10.1038/nphoton.2015.83.

[95] M. Peev et al. “The SECOQC quantum key distribution network in Vienna”. *New Journal of Physics* 11.7 (2009), p. 075001. ISSN: 1367-2630. DOI: 10.1088/1367-2630/11/7/075001.

[96] F. Xu et al. “Field experiment on a robust hierarchical metropolitan quantum cryptography network”. *Chinese Science Bulletin* 54.17 (2009), pp. 2991–2997. ISSN: 1001-6538. DOI: 10.1007/s11434-009-0526-3.

[97] M. Sasaki et al. “Field test of quantum key distribution in the Tokyo QKD Network”. *Optics Express* 19.11 (2011), p. 10387. ISSN: 1094-4087. DOI: 10.1364/OE.19.010387.

[98] M. Żukowski et al. ““Event-ready-detectors” Bell experiment via entanglement swapping”. *Physical Review Letters* 71.26 (1993), pp. 4287–4290. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.71.4287.

[99] W. J. Munro et al. “Quantum communication without the necessity of quantum memories”. *Nature Photonics* 6.11 (2012), pp. 777–781. ISSN: 1749-4885. DOI: 10.1038/nphoton.2012.243.

[100] M. Minder et al. “Experimental quantum key distribution beyond the repeaterless secret key capacity”. *Nature Photonics* 13.5 (2019), pp. 334–338. ISSN: 1749-4885. DOI: 10.1038/s41566-019-0377-7.

[101] Y. Tsunoo et al. “Cryptanalysis of DES Implemented on Computers with Cache”. *Cryptographic Hardware and Embedded Systems - CHES 2003*. Springer Berlin Heidelberg, 2003, pp. 62–76. ISBN: 978-3-540-45238-6. DOI: 10.1007/978-3-540-45238-6_6.

[102] P. C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. *Advances in Cryptology — CRYPTO ’96*. Springer Berlin Heidelberg, 1996, pp. 104–113. ISBN: 978-3-540-68697-2. DOI: 10.1007/3-540-68697-5_9.

[103] F.-X. Standaert et al. “A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks”. *Advances in Cryptology - EUROCRYPT 2009*. Springer Berlin Heidelberg, 2009, pp. 443–461. ISBN: 978-3-642-01001-9. DOI: 10.1007/978-3-642-01001-9_26.

[104] S. Chari et al. “Template Attacks”. *Cryptographic Hardware and Embedded Systems - CHES 2002*. Springer Berlin Heidelberg, 2003, pp. 13–28. ISBN: 978-3-540-36400-9. DOI: 10.1007/3-540-36400-5_3.

[105] W.-Y. Hwang. “Quantum Key Distribution with High Loss: Toward Global Secure Communication”. *Physical Review Letters* 91.5 (2003), p. 057901. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.91.057901.

[106] C.-H. F. Fung et al. “Phase-remapping attack in practical quantum-key-distribution systems”. *Physical Review A* 75.3 (2007), p. 032314. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.75.032314.

[107] S.-H. Sun et al. “Effect of source tampering in the security of quantum cryptography”. *Physical Review A* 92.2 (2015), p. 022304. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.92.022304.

[108] V. Makarov * and D. R. Hjelme. “Faked states attack on quantum cryptosystems”. *Journal of Modern Optics* 52.5 (2005), pp. 691–705. ISSN: 0950-0340. DOI: 10.1080/09500340410001730986.

[109] V. Makarov et al. “Effects of detector efficiency mismatch on security of quantum cryptosystems”. *Physical Review A* 74.2 (2006), p. 022313. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.74.022313.

[110] N. Jain et al. “Device Calibration Impacts Security of Quantum Key Distribution”. *Physical Review Letters* 107.11 (2011), p. 110501. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.107.110501.

[111] B. Qi et al. “Time-shift attack in practical quantum cryptosystems”. *Quantum Information and Computation* 7.1&2 (2007), pp. 73–82. ISSN: 15337146. DOI: 10.26421/QIC7.1-2-3.

[112] K. Wei et al. “Implementation security of quantum key distribution due to polarization-dependent efficiency mismatch”. *Physical Review A* 100.2 (2019), p. 022325. ISSN: 2469-9926. DOI: 10.1103/PhysRevA.100.022325.

[113] L. Lydersen et al. “Hacking commercial quantum cryptography systems by tailored bright illumination”. *Nature Photonics* 4.10 (2010), pp. 686–689. ISSN: 1749-4885. DOI: 10.1038/nphoton.2010.214.

[114] L. Lydersen et al. “Controlling a superconducting nanowire single-photon detector using tailored bright illumination”. *New Journal of Physics* 13.11 (2011), p. 113042. ISSN: 1367-2630. DOI: 10.1088/1367-2630/13/11/113042.

[115] C. Wiechers et al. “After-gate attack on a quantum cryptosystem”. *New Journal of Physics* 13.1 (2011), p. 013043. ISSN: 1367-2630. DOI: 10.1088/1367-2630/13/1/013043.

[116] L. Lydersen et al. “Superlinear threshold detectors in quantum cryptography”. *Physical Review A* 84.3 (2011), p. 032320. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.84.032320.

[117] A. N. Bugge et al. “Laser Damage Helps the Eavesdropper in Quantum Cryptography”. *Physical Review Letters* 112.7 (2014), p. 070503. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.112.070503.

[118] D. J. Rogers et al. “Detector dead-time effects and paralyzability in high-speed quantum key distribution”. *New Journal of Physics* 9.9 (2007), pp. 319–319. ISSN: 1367-2630. DOI: 10.1088/1367-2630/9/9/319.

[119] C. Kurtsiefer et al. “The breakdown flash of silicon avalanche photodiodes-back door for eavesdropper attacks?” *Journal of Modern Optics* 48.13 (2001), pp. 2039–2047. ISSN: 0950-0340. DOI: 10.1080/09500340108240905.

[120] H.-W. Li et al. “Attacking a practical quantum-key-distribution system with wavelength-dependent beam-splitter and multiwavelength sources”. *Physical Review A* 84.6 (2011), p. 062308. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.84.062308.

[121] N. Gisin et al. “Trojan-horse attacks on quantum-key-distribution systems”. *Physical Review A* 73.2 (2006), p. 022320. ISSN: 1050-2947. DOI: 10.1103/PhysRevA.73.022320.

[122] A. Vakhitov et al. “Large pulse attack as a method of conventional optical eavesdropping in quantum cryptography”. *Journal of Modern Optics* 48.13 (2001), pp. 2023–2038. ISSN: 0950-0340. DOI: 10.1080/09500340108240904.

[123] Y. Liu et al. “Experimental Measurement-Device-Independent Quantum Key Distribution”. *Physical Review Letters* 111.13 (2013), p. 130502. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.111.130502.

[124] D. P. Nadlinger et al. “Experimental quantum key distribution certified by Bell’s theorem”. *Nature* 607.7920 (2022), pp. 682–686. ISSN: 0028-0836. DOI: 10.1038/s41586-022-04941-5.

[125] B. Ahn et al. “Implementation of Plug & Play Quantum Key Distribution Protocol”. *Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2018, pp. 47–49. ISBN: 978-1-5386-4646-5. DOI: 10.1109/ICUFN.2018.8436633.

[126] *Laser and Quantum Optics group (LQO) at Technische Universität Darmstadt: Open-source software for controlling the quantum key distribution and its postprocessing*. URL: https://git.rwth-aachen.de/oleg.nikiforov/qkd-tools (visited on 04/03/2022).

[127] O. Nikiforov et al. *Side-Channel Analysis of Privacy Amplification in Postprocessing Software for a Quantum Key Distribution System*. Tech. rep. TUD-CS-2018-0024. Technische Universität Darmstadt, 2018. URL: https://www.mais.informatik.tu-darmstadt.de/qkd-esorics21.html.

[128] K. P. Murphy. *Machine learning: A Probabilistic Perspective*. MIT Press, 2012. ISBN: 978-0-262-01802-9.

[129] M. J. Zaki and W. Meira. *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. 2nd. Cambridge University Press, 2020. ISBN: 9781108473989.

[130] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[131] C. M. Bishop. *Pattern Recognition and Machine Learning*. 1st. Springer New York, 2006. ISBN: 978-0387-31073-2.

[132] E. Fix and J. L. Hodges. “Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties”. *International Statistical Review / Revue Internationale de Statistique* 57.3 (1989), p. 238. ISSN: 03067734. DOI: 10.2307/1403797.

[133] T. Cover and P. Hart. “Nearest neighbor pattern classification”. *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. ISSN: 0018-9448. DOI: 10.1109/TIT.1967.1053964.

[134] P. Domingos and M. Pazzani. “On the Optimality of the Simple Bayesian Classifier under Zero-One Loss”. *Machine Learning* 29.2/3 (1997), pp. 103–130. ISSN: 08856125. DOI: 10.1023/A:1007413511361.

[135] A. Niculescu-Mizil and R. Caruana. “Predicting good probabilities with supervised learning”. *Proceedings of the 22nd international conference on Machine learning - ICML ’05*. ACM Press, 2005, pp. 625–632. ISBN: 1595931805. DOI: 10.1145/1102351.1102430.

[136] B. E. Boser et al. “A training algorithm for optimal margin classifiers”. *Proceedings of the fifth annual workshop on Computational learning theory - COLT ’92*. ACM Press, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401.

[137] V. N. Vapnik and A. Y. Chervonenkis. “On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities”. *Theory of Probability & Its Applications* 16.2 (1971), pp. 264–280. ISSN: 0040-585X. DOI: 10.1137/1116025.

[138] E. J. Nyström. “Über Die Praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben”. (in German). *Acta Mathematica* 54 (1930), pp. 185–204. ISSN: 1871-2509. DOI: 10.1007/BF02547521.

[139] J. Platt. “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods”. *Advances in Large Margin Classifiers* 10.3 (1999), pp. 61–74.

[140] L. Hyafil and R. L. Rivest. “Constructing optimal binary decision trees is NP-complete”. *Information Processing Letters* 5.1 (1976), pp. 15–17. ISSN: 00200190. DOI: 10.1016/0020-0190(76)90095-8.

[141] L. Breiman. “Bagging predictors”. *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 0885-6125. DOI: 10.1007/BF00058655.

[142] L. Breiman. “Random Forests”. *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 08856125. DOI: 10.1023/A:1010933404324.

[143] T. M. Oshiro et al. “How Many Trees in a Random Forest?” *Machine Learning and Data Mining in Pattern Recognition*. Springer Berlin Heidelberg, 2012, pp. 154–168. ISBN: 978-3-642-31537-4. DOI: 10.1007/978-3-642-31537-4_13.

[144] P. Probst and A.-L. Boulesteix. “To Tune or Not to Tune the Number of Trees in Random Forest”. *Journal of Machine Learning Research* 18.181 (2018), pp. 1–18. URL: http://jmlr.org/papers/v18/17-269.html.

[145] R. E. Schapire. “The strength of weak learnability”. *Machine Learning* 5.2 (1990), pp. 197–227. ISSN: 0885-6125. DOI: 10.1007/BF00116037.

[146] A. J. Wyner et al. “Explaining the Success of AdaBoost and Random Forests as Interpolating Classifiers”. *Journal of Machine Learning Research* 18.48 (2017), pp. 1–33. URL: http://jmlr.org/papers/v18/15-240.html.

[147] Y. Freund and R. E. Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 00220000. DOI: 10.1006/jcss.1997.1504.

[148] L. Breiman. *Arcing the Edge*. Tech. rep. 486. University of California, Berkeley, 1997. URL: https://statistics.berkeley.edu/tech-reports/486.

[149] J. H. Friedman. “Greedy function approximation: A gradient boosting machine.” *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 0090-5364. DOI: 10.1214/aos/1013203451.

[150] T. Hastie et al. “Boosting and Additive Trees”. *The Elements of Statistical Learning*. 1st. Springer New York, 2001, pp. 299–345. ISBN: 978-0-387-21606-5. DOI: 10.1007/978-0-387-21606-5_10.

[151] J. H. Friedman. “Stochastic gradient boosting”. *Computational Statistics & Data Analysis* 38.4 (2002), pp. 367–378. ISSN: 01679473. DOI: 10.1016/S0167-9473(01)00065-2.

[152] M. Feurer and F. Hutter. “Hyperparameter Optimization”. *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing, 2019, pp. 3–33. ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_1.

[153] J. Bergstra and Y. Bengio. “Random Search for Hyper-Parameter Optimization”. *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305. URL: http://jmlr.org/papers/v13/bergstra12a.html.

[154] J. Petrak. *Fast Subsampling Performance Estimates for Classification Algorithm Selection*. Technical Report TR-2000-07. Austrian Research Institute for Artificial Intelligence (OFAI), 2000. URL: https://ofai.at/papers/oefai-tr-2000-07.pdf.

[155] A. van den Bosch. “Wrapped progressive sampling search for optimizing learning algorithm parameters”. *Proceedings of the Belgium-Netherlands Conference on Artificial Intelligence, BNAIC’04*. 2004, pp. 219–226. URL: https://www.ai.rug.nl/conf/bnaic2004/ap/a9.pdf.

[156] C. Thornton et al. “Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms”. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855. ISBN: 9781450321747. DOI: 10.1145/2487575.2487629.

[157] T. Krueger et al. “Fast Cross-Validation via Sequential Testing”. *Journal of Machine Learning Research* 16.33 (2015), pp. 1103–1155. URL: http://jmlr.org/papers/v16/krueger15a.html.

[158] E. R. Sparks et al. “Automating model search for large scale machine learning”. *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 368–380. ISBN: 9781450336512. DOI: 10.1145/2806777.2806945.

[159] A. Sabharwal et al. “Selecting Near-Optimal Learners via Incremental Data Allocation”. *AAAI Conference on Artificial Intelligence*. AAAI’16. AAAI Press, 2016, pp. 2007–2015. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12524.

[160] Z. Karnin et al. “Almost Optimal Exploration in Multi-Armed Bandits”. *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. Proceedings of Machine Learning Research 3. PMLR, 2013, pp. 1238–1246. URL: https://proceedings.mlr.press/v28/karnin13.html.

[161] K. Jamieson and A. Talwalkar. “Non-stochastic Best Arm Identification and Hyperparameter Optimization”. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Vol. 51. Proceedings of Machine Learning Research. PMLR, 2016, pp. 240–248. URL: https://proceedings.mlr.press/v51/jamieson16.html.

[162] L. Li et al. “A System for Massively Parallel Hyperparameter Tuning”. *Proceedings of Machine Learning and Systems*. Vol. 2. 2020, pp. 230–246. URL: https://proceedings.mlsys.org/paper/2020/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html.

[163] A. Wiltgen et al. “Power consumption analysis in static CMOS gates”. *26th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 2013, pp. 1–6. ISBN: 978-1-4799-1132-5. DOI: 10.1109/SBCCI.2013.6644863.

[164] M. M. Mano et al. *Logic and Computer Design Fundamentals*. 5th. Pearson Education, 2015. ISBN: 9780134080154.

[165] C. D. Motchenbacher and J. A. Connelly. *Low-Noise Electronic System Design*. 1st. Wiley Interscience, 1993. ISBN: 9780471577423.

[166] A. Shamir. “How to share a secret”. *Communications of the ACM* 22.11 (1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176.

[167] E. Prouff and M. Rivain. “Masking against Side-Channel Attacks: A Formal Security Proof”. *Advances in Cryptology – EUROCRYPT 2013*. Springer Berlin Heidelberg, 2013, pp. 142–159. ISBN: 978-3-642-38348-9. DOI: 10.1007/978-3-642-38348-9_9.

[168] J. F. Bravo and P. Mateus. “Side-channel Analysis of Quantum Key Distribution: Privacy Amplification Power-traces”. *Entropy* 24.12 (2022). (to be published).

[169] D. C. Liu and J. Nocedal. “On the limited memory BFGS method for large scale optimization”. *Mathematical Programming* 45.1-3 (1989), pp. 503–528. ISSN: 0025-5610. DOI: 10.1007/BF01589116.

[170] A. Defazio et al. “SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives”. *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/ede7e2b6d13a41ddf9f4bdef84fdc737-Paper.pdf.

[171] C. Cortes and V. Vapnik. “Support-vector networks”. *Machine Learning* 20.3 (1995), pp. 273–297. ISSN: 0885-6125. DOI: 10.1007/BF00994018.

[172] S. Shalev-Shwartz et al. “Pegasos: primal estimated sub-gradient solver for SVM”. *Mathematical Programming* 127.1 (2011), pp. 3–30. ISSN: 0025-5610. DOI: 10.1007/s10107-010-0420-4.

[173] J. Platt. “Fast Training of Support Vector Machines Using Sequential Minimal Optimization”. *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998, pp. 185–208. URL: https://www.microsoft.com/en-us/research/publication/fast-training-of-support-vector-machines-using-sequential-minimal-optimization.

[174] L. Breiman et al. *Classification And Regression Trees*. 1st. Chapman & Hall, 1984. ISBN: 9780412048418. DOI: 10.1201/9781315139470.

[175] J. R. Quinlan. “Induction of decision trees”. *Machine Learning* 1.1 (1986), pp. 81–106. ISSN: 0885-6125. DOI: 10.1007/BF00116251.

[176] J. R. Quinlan. *C4.5: Programs for Machine Learning*. 1st. Morgan Kaufmann Publishers, 1993. ISBN: 978-1558602380.

[177] M. Kuhn and K. Johnson. *Applied Predictive Modeling*. Springer New York, 2013. ISBN: 978-1-4614-6848-6. DOI: 10.1007/978-1-4614-6849-3.

[178] L. Mason et al. “Boosting Algorithms as Gradient Descent”. *Advances in Neural Information Processing Systems, NIPS’99*. Vol. 12. MIT Press, 1999, pp. 512–518. URL: http://papers.nips.cc/paper/1766-boosting-algorithms-as-gradient-descent.

Appendix A

Machine-learning models

In this appendix, we provide a mathematical description of the models used in our analysis. We also describe their hyperparameters and how they influence each model. For each model, we assume we have the input features x_1, x_2, ..., x_n and a target y which takes values in $\{0, 1\}$. We focus on the binary-target case, since we will only deal with that type of problems.

A.1 Logistic regression

In a logistic regression [128], given an input vector $\boldsymbol{x}$ with $x_0 = 1$ for convenience and the remaining entries as the input features, the probability of the target being 1 is assumed to be of the form

$$P(y = 1|\boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T \cdot \boldsymbol{x}\right) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right), \tag{A.1}$$

where σ is the logistic function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{A.2}$$

$\boldsymbol{w}$ is called the weight vector and w_0 is usually denoted as the intercept or bias term.

Then, it follows that the probability of the target being 0 is simply $P(y = 0|\boldsymbol{x}) = 1 - P(y = 1|\boldsymbol{x})$.

Training the logistic regression with N records is an optimization problem of finding the best weight vector for minimizing the a loss function. Specifically, it is

$$\min_{\boldsymbol{w}} \sum_{l=1}^{N} \Big(- y^l \log\big(P(y^l = 1|\boldsymbol{x}^l)\big) - (1 - y^l) \log\big(P(y^l = 0|\boldsymbol{x}^l)\big)\Big) + \lambda r(\boldsymbol{w}), \tag{A.3}$$

where λ is a regularization constant, $\boldsymbol{x}^l$ and y^l are the input vector and the real output of record l, respectively, and $r(\boldsymbol{w})$ is a regularization term.

Setting $r(\boldsymbol{w}) = 0$ amounts to no regularization. In that case, the optimization problem reduces to minimizing the cross-entropy loss, also known as log loss, between the true target values and their predicted probabilities. This loss function takes the form

$$H(p,q) = -\sum_i p_i \log(q_i), \tag{A.4}$$

where p and q are two distributions.

Introducing regularization helps prevent overfitting as described in section 2.4.4. A regularization term imposes an additional penalty on the loss function, which also improves the numerical stability of the optimization algorithm. In the optimization problem (A.3), the constant λ controls the amount of regularization, with greater values corresponding to higher regularization. The forms of regularization most commonly used in machine learning are L1-norm regularization, in which $r(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_{i=0}^{n} |w_i|$, and L2-norm regularization, in which $r(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 = \frac{1}{2}\sum_{i=0}^{n} w_i^2$ [130].

In practice, there are multiple ways of solving the optimization problem (A.3). Usually the loss function does not have a closed-form solution, so iterative optimization solvers are used. Common ones are the L-BFGS [169], which applies a quasi-Newton method, and SAGA, which is a stochastic average-gradient-descent method [170].

A.2 k-nearest neighbours

In kNN classification, a common approach to determining the target of a new record is majority voting of the other k targets [130], which corresponds to averaging the targets and applying a 0.5 threshold to the outcome. Another approach is performing the weighted average of the targets according to the inverse of the distance of the training record to the new record, and applying the same 0.5 threshold to the outcome. The latter approach attributes greater weights to closer records. With both approaches, k is recommended to be odd to avoid ties.

The most-commonly used distance metrics are the Manhattan distance, $d_1(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|_1$, the Euclidean distance, $d_2(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|_2$, and the Chebyshev distance, $d_\infty = \|\boldsymbol{x}-\boldsymbol{y}\|_\infty = \max_i(|x_i - y_i|)$ [130].

A.3 Naïve Bayes

Naïve Bayes methods "naively" assume conditional independence between every pair of input features, given the target value [128]. Mathematically, they assume

$$P(x_i|y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|y). \tag{A.5}$$

For a vector of input features $\boldsymbol{x}$, this allows us to write

$$P(\boldsymbol{x}|y) = \prod_{i=1}^{n} P(x_i|y). \tag{A.6}$$

Then, we can write Bayes' theorem as

$$P(y|\boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x}|y)}{P(\boldsymbol{x})} = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(\boldsymbol{x})} \quad \equiv \quad \text{Posterior} = \frac{\text{Prior} \times \text{Likelihood}}{\text{Evidence}}. \tag{A.7}$$

Since the evidence is a constant for a specific input vector $\boldsymbol{x}$, only the numerator will determine the predicted target value $\hat{y}$ of a new record. Therefore, using

$$P(y|\boldsymbol{x}) \propto P(y) \prod_{i=1}^{n} P(x_i|y), \tag{A.8}$$

for binary-target problems, naïve Bayes methods use the classification rule

$$\hat{y} = \underset{y \in \{0,1\}}{\arg\max}\, P(y|\boldsymbol{x}) = \underset{y \in \{0,1\}}{\arg\max}\, P(y) \prod_{i=1}^{n} P(x_i|y). \tag{A.9}$$

The priors can either be assumed to be uniform or estimated based on the target-value probabilities, as $P(y) = \frac{N_y}{N}$, where N_y is the number of training records with target value y and N is the total number of training records. The former assumption results in a maximum-likelihood estimation (MLE) problem and the latter results in a maximum a-posteriori (MAP) estimation problem [128].

In the Gaussian naïve Bayes, we assume the likelihood to be a product of Gaussian distributions [128] or, equivalently,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_{i,y}^2}} \exp\left(-\frac{(x_i - \mu_{i,y})^2}{2\sigma_{i,y}^2}\right), \tag{A.10}$$

where $\mu_{i,y}$ is the mean of feature x_i for records with target y and $\sigma_{i,y}^2$ is its variance. In practice, a smoothing factor α can be added to the variances as a way of introducing regularization into the model and stabilizing calculations.

As mentioned in section 2.4.3, the multinomial naïve Bayes is most-commonly used in text-classification tasks, with each feature being the number of times specific words appear in a text. It assumes a multinomial likelihood distribution [128],

$$P(\boldsymbol{x}|y) = \frac{\left(\sum_{i=1}^{n} x_i\right)!}{\prod_{i=1}^{n} x_i!} \prod_{i=1}^{n} \left(P_{i,y}\right)^{x_i}. \tag{A.11}$$

In the text-classification example, $P_{i,y}$ is the probability of the word corresponding to feature x_i being found in records with target y. More generally, it is the sum of the values the feature x_i takes when appearing in records with target y divided by the sum of all values of all features in those records,

$$P_{i,y} = \frac{\sum_{X_y} x_i}{\sum_{X_y} \sum_{j=1}^{n} x_j}, \tag{A.12}$$

where X_y is the subset of records with target y.

Again, in text classification, when a new record has a feature x_i the model has not seen before, $P_{i,y}$ becomes zero. This would make $P(\boldsymbol{x}|y) = 0$, erasing all information about the other features. In order to account for feature values not present in the training data, $P_{i,y}$ is computed as [130]

$$P_{i,y} = \frac{\sum_{X_y} x_i + \alpha}{\sum_{X_y} \sum_{j=1}^{n} x_j + \alpha n}, \tag{A.13}$$

where n is the number of features and α is the smoothing factor which prevents null probabilities. When $\alpha = 1$, this is known as Laplace smoothing and, in the general case, it is known as Lidstone smoothing [128].

It is interesting to look at this problem from another perspective. Since the terms with factorials are constant for a specific input vector $\boldsymbol{x}$, we have

$$P(y|\boldsymbol{x}) \propto P(y) \prod_{i=1}^{n} P_{i,y}^{x_i} \tag{A.14}$$

$$\implies \log(P(y|\boldsymbol{x})) \propto \log(P(y)) + \sum_{i=1}^{n} \log(P_{i,y})\, x_i \equiv w_0 + \boldsymbol{w_y^T} \cdot \boldsymbol{x}, \tag{A.15}$$

where $w_0 = \log(P(y))$ and $w_{i,y} = \log(P_{i,y})$. As we can see, the multinomial naïve Bayes becomes a linear model when we want to find the logarithm of the posterior probabilities.

Finally, the categorical naïve Bayes assumes the likelihood to be a product of categorical or multinoulli distributions [128]. If each input feature x_i has k_i categories, let $j_i \in \{1, ..., k_i\}$ represent a category of x_i. Then, the probability of the vector of input features $\boldsymbol{x}$ being equal to the vector of categories $\boldsymbol{j} = (j_1, ..., j_n)$, given the target value y, is

$$P(\boldsymbol{x} = \boldsymbol{j}|y) = \prod_{i=1}^{n} P(x_i = j_i|y), \tag{A.16}$$

where $P(x_i = j_i|y)$ is the probability of feature x_i having the category j_i in the subset of records with target y. In practice, this probability is computed as [130]

$$P(x_i = j_i|y) = \frac{N_{i,j_i,y} + \alpha}{N_y + \alpha k_i}. \tag{A.17}$$

$N_{i,j_i,y}$ is the cardinality of the subset $\left\{l \in \{1, ..., N\} : x_i^l = j_i, y^l = y\right\}$, with N being the number of training records, x_i^l the feature x_i of record l and y^l the output of that record. Then, $N_{i,j_i,y}$ is the number of times the category j_i appears in the feature x_i of the records with target y. N_y is the cardinality of the subset $\left\{l \in \{1, ..., N\} : y^l = y\right\}$, which is the number of records with target y. The smoothing factor α helps account for new categories and prevent null probabilities, just like in the multinomial case.

A.4 Support-vector machine

Let us consider a target feature y which takes values in $\{-1, 1\}$ to simplify the notation. To build a SVM with n input features, we want to find the $(n-1)$-dimensional hyperplane [136]

$$\boldsymbol{w}^T \cdot \boldsymbol{x} + w_0 = 0, \tag{A.18}$$

where $\boldsymbol{x}$ is a vector of input features and parameters $\boldsymbol{w}$ and w_0 are such that

$$y^l \left(\boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0\right) > 0, \quad \forall\, l \in \{1, ..., N\}, \tag{A.19}$$

where N is the number of training records. In words, we want to find a hyperplane which separates the training records according to their target value. However, for some linearly-separable problems, multiple such hyperplanes accomplish that task exactly. In order to build a model with the best generalization performance possible, we choose the hyperplane which maximizes the margin. Since $\boldsymbol{w}$ is orthogonal to the hyperplane, the margin m is defined as

$$m(\boldsymbol{w}, w_0) = \min_l \frac{y^l \left(\boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0\right)}{\|\boldsymbol{w}\|_2}, \tag{A.20}$$

where the expression to be minimized is the distance of data point $\boldsymbol{x}^l$ to the hyperplane, and the data points which are the solution to this minimization are the support vectors. Then, finding the optimal hyperplane is equivalent to solving the constrained optimization problem

$$\underset{\boldsymbol{w}, w_0}{\arg\max}\; m(\boldsymbol{w}, w_0) \quad \text{subject to} \quad y^l(\boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0) > 0, \quad \forall\, l \in \{1, ..., N\}. \tag{A.21}$$

Since the rescaling of the parameters $\boldsymbol{w}$ and w_0 by any constant does not change the distance of any data point x^l to the hyperplane, it is common to consider a scaling such that we can use the canonical representation of the hyperplane

$$y^l(\boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0) \geq 1, \quad \forall\, l \in \{1, ..., N\}. \tag{A.22}$$

The support vectors lead to an equality in the inequation above. Thus, the constrained optimization problem (A.21) reduces to

$$\underset{\boldsymbol{w}, w_0}{\arg\min}\; \|\boldsymbol{w}\|_2^2 \quad \text{subject to} \quad y^l(\boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0) \geq 1, \quad \forall\, l \in \{1, ..., N\}, \tag{A.23}$$

where we use the square of the norm for computational convenience.

In non-linearly-separable problems, there will be no solution to the optimization problem (A.23). Therefore, a slack variable $\xi^l \geq 0$ is introduced for each training data point [171]. If the data point x^l is on the correct side of the hyperplane and on or outside the margin boundary, then $\xi^l = 0$. Otherwise, $\xi^l = \left|y^l - \boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0\right|$. If $0 < \xi^l \leq 1$, the data point is on the correct side of the hyperplane but inside the margin. If $\xi^l > 1$, it is on the wrong side of the hyperplane. Thus, the slack variables act as penalties which increase with the distance of the data points to the correct side of the margin boundary. It can be shown [171] that the optimization problem becomes

$$\underset{\boldsymbol{w}, w_0}{\arg\min}\; \|\boldsymbol{w}\|_2^2 + \lambda \sum_{l=1}^{N} \xi^l \quad \text{subject to} \quad \begin{cases} y^l(\boldsymbol{w}^T \cdot \boldsymbol{x}^l + w_0) \geq 1 - \xi^l \\ \xi^l \geq 0 \end{cases}, \quad \forall\, l \in \{1, ..., N\}, \tag{A.24}$$

with $\lambda > 0$ acting as a regularization parameter which controls the number of training mistakes we are willing to tolerate. The lower its value, the more tolerable mistakes and the higher the regularization. This is known as the soft-margin approach.

The constrained optimization problem (A.24) can be solved directly by methods such as sub-gradient descent [172], or indirectly by techniques such as using the Lagrange-multiplier method to obtain the simpler dual optimization problem as shown in [171]. The dual problem can then be solved with techniques such as sequential minimal optimization (SMO) [173].

After determining $\boldsymbol{w}$ and w_0 and therefore knowing the optimal hyperplane, a new record is classified with

$$\hat{y} = \text{sgn}\left(\boldsymbol{w}^T \cdot \boldsymbol{x} + w_0\right), \tag{A.25}$$

where sgn is the sign function

$$\text{sgn}(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}. \tag{A.26}$$

The method detailed above builds a linear model. In order to perform non-linear classification, we must map the input features into a higher-dimensional space [136]. This is done using a kernel function, which, as mentioned in section 2.4.3, takes the form

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}'), \tag{A.27}$$

with $\boldsymbol{\phi}(\boldsymbol{x})$ being a specific feature-space mapping. By first transforming the input-feature space and then applying the linear SVM algorithm, equation (A.25) becomes

$$\hat{y} = \text{sgn}\left(\boldsymbol{w}^T \cdot \boldsymbol{\phi}(\boldsymbol{x}) + w_0\right). \tag{A.28}$$

This is a linear model in the transformed input-feature space, but a non-linear one in the original input-feature space.

Factorizing a kernel function can be quite difficult. In order to avoid doing that by working in the input-feature space, we can work in the dual space. This can be achieved by applying the Lagrange-multiplier method to the constrained optimization problem (A.24), as stated above. Then, it can be shown [136] that the predicted target $\hat{y}$ for a new record $\boldsymbol{x}$ takes the form

$$\hat{y} = \sum_{l=1}^{N} \alpha^l y^l k(\boldsymbol{x}, \boldsymbol{x}^l) + w_0 \tag{A.29}$$

where α^l is the Lagrange multiplier associated with the first constraint for the record l and k is the chosen kernel function. This is known as the kernel trick. It can also be shown [136] that $0 \leq \alpha^l \leq \lambda$, with $\alpha^l = 0$ if $\boldsymbol{x}^l$ is not a support vector, which is congruent with the assertion that only support vectors are needed to build a SVM.

As mentioned in section 2.4.3, besides the linear SVM, where $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \cdot \boldsymbol{x}'$, commonly-used non-linear kernel functions are the gaussian radial-basis function (RBF), polynomial functions and the sigmoid function [128].

A radial-basis function is a function which fulfils $f(\boldsymbol{x}) = f(\|\boldsymbol{x}\|)$. The Gaussian RBF kernel function is given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\Big(-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2\Big), \quad \gamma > 0. \tag{A.30}$$

The d-th-degree polynomial kernel function takes the form

$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(\gamma \boldsymbol{x}^T \cdot \boldsymbol{x}' + b\right)^d, \quad \gamma, b \in \mathbb{R}. \tag{A.31}$$

The sigmoid kernel function is

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh\big(\gamma \boldsymbol{x}^T \cdot \boldsymbol{x}' + b\big), \quad \gamma, b \in \mathbb{R}. \tag{A.32}$$

As mentioned before, a proper choice of kernel function and regularization parameter λ is crucial to prevent overfitting.

A.5 Decision tree

Training a decision tree is done with greedy algorithms which perform locally-optimal splits of the input-feature space [128]. These splits are represented by decision nodes in a diagram.

Let $\mathcal{N}_j$ be the subset of N_j records corresponding to node j. A candidate split (x_i, t_j), where x_i is the chosen input feature and t_j is the chosen splitting threshold, partitions $\mathcal{N}_j$ into $\mathcal{N}_j^{left}$ and $\mathcal{N}_j^{right}$ such that

$$\mathcal{N}_j^{left}(x_i, t_j) = \{(\boldsymbol{x}, y) : x_i \leq t_j\} \tag{A.33}$$

$$\mathcal{N}_j^{right}(x_i, t_j) = \mathcal{N}_j \setminus \mathcal{N}_j^{left}(x_i, t_j) \tag{A.34}$$

where $\boldsymbol{x} = (x_1, ..., x_n)$. In order to determine the optimal split at node j, we define a loss function L which is the weighted sum of a loss function H on each of the resulting subsets,

$$L(\mathcal{N}_j, x_i, t_j) = \frac{N_j^{left}}{N_j} H\left(\mathcal{N}_j^{left}(x_i, t_j)\right) + \frac{N_j^{right}}{N_j} H\left(\mathcal{N}_j^{right}(x_i, t_j)\right), \tag{A.35}$$

where N_j^{left} and N_j^{right} are the number of records in subsets $\mathcal{N}_j^{left}$ and $\mathcal{N}_j^{left}$, respectively. Then, the optimal split is the solution to the optimization problem

$$\underset{x_i, t_j}{\arg\min}\ L(\mathcal{N}_j, x_i, t_j). \tag{A.36}$$

The fraction of records with target value y at node j is

$$P_{j,y} = \frac{1}{N_j} \sum_{l \in \mathcal{N}_j} \mathbb{1}(y^l = y). \tag{A.37}$$

For a binary classification problem, commonly-used loss functions H are the Gini impurity, or Gini index,

$$\begin{aligned} H(\mathcal{N}_j) &= P_{j,0}\left(1 - P_{j,0}\right) + P_{j,1}\left(1 - P_{j,1}\right) \\ &= 1 - P_{j,0}^2 - P_{j,1}^2 \end{aligned} \tag{A.38}$$

and the binary Shannon entropy

$$H(\mathcal{N}_j) = -P_{j,0}\log(P_{j,0}) - P_{j,1}\log(P_{j,1}). \tag{A.39}$$

The binary Shannon entropy penalizes solutions containing subsets with low purity more than the Gini impurity. As mentioned in section 2.4.3, the prediction of the final model is either the most common target value of the records in each leaf or the probability of a record classified into leaf j having target value y, which is given by $P_{j,y}$.

For a regression problem with a continuous target variable y, it is common to use either the mean squared error (MSE) or the mean absolute error (MAE) as the loss function H [130]. The MSE loss function is given by

$$H(\mathcal{N}_j) = \frac{1}{N_j}\sum_{l\in\mathcal{N}_j}\left(y^l - \overline{y}_j\right)^2, \tag{A.40}$$

where $\overline{y}_j$ is the mean target value of the records in $\mathcal{N}_j$. The MAE loss function is

$$H(\mathcal{N}_j) = \frac{1}{N_j}\sum_{l\in\mathcal{N}_j}\left|y^l - \tilde{y}_j\right|, \tag{A.41}$$

where $\tilde{y}_j$ is the median target value of the records in $\mathcal{N}_j$. When using the MSE loss function, the final predicted value for leaf j is $\overline{y}_j$ and, when using MAE, the final prediction is $\tilde{y}_j$.

As mentioned in section 2.4.3, multiple pre-pruning strategies are commonly used to improve a decision tree's generalization ability, one of which is to define the minimum reduction in impurity allowed for a split to happen. The reduction in impurity associated with a split at node j is defined as [130]

$$R(\mathcal{N}_j, x_i, t_j) = \frac{N_j}{N}\left(H(\mathcal{N}_j) - L(\mathcal{N}_j, x_i, t_j)\right), \tag{A.42}$$

where N is the total number of training records.

Common algorithms which implement a greedy approach similar to the one described above, with pre- and post-pruning strategies, are CART [174], ID3 [175], C4.5 [176] and its successor C5.0 [177], which is faster than C4.5, more memory efficient, and creates smaller trees with the same performance results.

A.6 Random forest

As with any bagging model, the final prediction of a random forest is either given by a majority voting or an average of the outputs of the base decision trees [130].

A.7 Adaboost

In AdaBoost, M models are trained sequentially and the training records are reweighted as the training iterations progress [147].

In AdaBoost, the first model is trained on the original dataset, where every record has the same weight. Then, the weights are readjusted in such a way that wrongly-classified training records are given larger weights than correctly-predicted ones, and a new model is trained with that reweighted dataset. The same reweighting-and-training procedure is done for following models. This way, hard-to-predict records gain increasing influence in the model training as iterations progress. The final prediction of an AdaBoost model is an average of the M base-model predictions weighted according to the number of records wrongly predicted by each model.

Let w_m^l correspond to the weight of record l at iteration m. Then, for N training records, we start with

$$w_0^l = \frac{1}{N}, \quad \forall\, l \in 1, ..., N. \tag{A.43}$$

After training the first model, we compute its weighted error rate ϵ_1 on the training dataset with

$$\epsilon_m = \sum_{l=1}^{N} w_{m-1}^l \mathbb{1}\left(\hat{y}_m^l \neq y^l\right), \tag{A.44}$$

where $\hat{y}_m^l$ is the predicted target value for record l by the m-th model and y^l is its real target value. If $\epsilon_m = 0$, we stop the training iterations since the AdaBoost model has fully captured the underlying structure of the training dataset.

The weight associated to model m in the weighted average of the final prediction is

$$\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right). \tag{A.45}$$

As stated in section 2.4.3, models with larger error rates will have a smaller contribution to the final prediction. We proceed by updating the record weights with

$$w_m^l = w_{m-1}^l \exp\left(\eta\alpha_m \mathbb{1}\left(\hat{y}_m^l \neq y^l\right)\right), \quad \eta \in (0, 1], \tag{A.46}$$

where η is the user-specified learning rate. Effectively, this leaves the weights unchanged when a target value is correctly predicted, i.e, $\hat{y}_m^l = y^l$, and multiplies the weights of the wrongly-predicted records by $\exp(\eta\alpha_m) = \left(\frac{1-\epsilon_m}{\epsilon_m}\right)^\eta$. Thus, wrong predictions by good models, i.e., models with a low error rate, lead to large increases in the respective weights. On the other hand, when weak models make wrong predictions, the respective weights are only slightly increased since weak models are less trustworthy. If a model has $\epsilon_m \geq 0.5$, it is equal or worse than random guessing in the binary-target case. Since this would lead to unchanged or even smaller weights for the wrongly-predicted records, we discard this model and retrain a new base model.

After updating the record weights, we normalize them so that their sum is 1 and train the following base model with the reweighted dataset.

For the binary-classification case, the final prediction of an AdaBoost model is given by

$$P(y=1|\boldsymbol{x}) = \frac{\sum_{m=1}^{M} \alpha_m P_m(y=1|\boldsymbol{x})}{\sum_{m=1}^{M} \alpha_m} \tag{A.47}$$

if the m-th base model's output is the probability of record $\boldsymbol{x}$ having target value 1, $P_m(y=1|\boldsymbol{x})$. In case the base model's output is the predicted target class, we simply replace the probabilities in the sum of equation (A.47) with $\hat{y}^l_m$. In order to obtain a target-value prediction, we define a threshold probability as mentioned before.

A.8 Gradient-boosting machine

In a GBM, M regression base models are trained to predict the negative gradient of a loss function H for each record [149]. The base model $\mathcal{M}_m$ is trained and weighted by a factor β_m to minimize the total loss on the training dataset. This is translated into the optimization problem

$$\underset{\mathcal{M}_m,\beta_m}{\arg\min} \sum_{l=1}^{N} H\left(y^l, P_{m-1}(\boldsymbol{x}^l) + \beta_m \mathcal{M}_m(\boldsymbol{x}^l)\right) \tag{A.48}$$

for each base model, where $\mathcal{M}_m(\boldsymbol{x}^l)$ is the prediction of model $\mathcal{M}_m$ for record l and

$$P_{m-1}(\boldsymbol{x}^l) = P_0 + \sum_{j=1}^{m-1} \eta \beta_j \mathcal{M}_j(\boldsymbol{x}^l), \tag{A.49}$$

with $\eta \in (0,1]$ being the user-specified learning rate. Then, $P_{m-1}(\boldsymbol{x}^l)$ is the prediction for record l of the current base-model ensemble at iteration $m-1$. The initial value P_0 is chosen as the constant which minimizes the total loss function on the training dataset.

As mentioned in section 2.4.3, the learning rate controls the contribution of each base model by maintaining or decreasing its effective weight in the ensemble after the respective optimization. In turn, this affects the weights β_m chosen in following iterations.

The loss function H may be approximated to first order as

$$H\left(y^l, P_{m-1}(\boldsymbol{x}^l) + \beta_m \mathcal{M}_m(\boldsymbol{x}^l)\right) \approx H\left(y^l, P_{m-1}(\boldsymbol{x}^l)\right) + \beta_m \mathcal{M}_m(\boldsymbol{x}^l) \left.\frac{\partial H\left(y^l, P(\boldsymbol{x}^l)\right)}{\partial P(\boldsymbol{x}^l)}\right|_{P=P_{m-1}}. \tag{A.50}$$

By using this first-order approximation in equation (A.48) and removing the constant terms at iteration m, we have

$$\underset{\mathcal{M}_m,\beta_m}{\arg\min} \sum_{l=1}^{N} \beta_m \mathcal{M}_m(\boldsymbol{x}^l) \left.\frac{\partial H\left(y^l, P(\boldsymbol{x}^l)\right)}{\partial P(\boldsymbol{x}^l)}\right|_{P=P_{m-1}}. \tag{A.51}$$

As stated above, the solution to this optimization problem is a model which is trained to predict a value proportional to the symmetric of the partial derivative. Moreover, as the gradients are updated at each iteration, this is akin to a gradient descent in a functional space as was shown in [178].

Let us consider again a target feature y with values in $\{-1, 1\}$ to simplify the notation. Then, two commonly-used loss functions are the exponential loss [128]

$$H(y, P(\boldsymbol{x})) = \exp(-yP(\boldsymbol{x})) \tag{A.52}$$

and the log loss in the form

$$H(y, P(\boldsymbol{x})) = \log(1 + \exp(-yP(\boldsymbol{x}))). \tag{A.53}$$

The exponential loss attributes larger penalties to wrongly-predicted outputs than the log loss and is not generalizable to the multiple-target-value case while the log loss is.

For both loss functions, the initial value P_0 is

$$P_0 = \frac{1}{2}\log\left(\frac{\gamma}{1-\gamma}\right), \quad \text{with} \quad \gamma = \frac{1}{N}\sum_{l=1}^{N} \mathbb{1}(y^l = 1). \tag{A.54}$$

A stated in section 2.4.3, using the exponential loss function reduces a GBM to the AdaBoost model. However, we cannot recover reasonable probabilities from the gradient predictions for this loss [128]. Therefore, we directly obtain the predicted target value with

$$\hat{y} = \text{sgn}\left(P_M(\boldsymbol{x})\right). \tag{A.55}$$

A GBM using the log loss is also called LogitBoost model [149]. With this loss, we can recover probabilities from the base-model predictions with

$$P(y = 1|\boldsymbol{x}) = \sigma\left(P_M(\boldsymbol{x})\right), \tag{A.56}$$

where σ is the logistic function.

Appendix B

All model results

In this appendix, we present a summary of all model results from the three studied approaches.

B.1 1-bit-model results

For the 1-bit models, their accuracy for each combination of noise level and sampling interval is presented in tables B.1 to B.3, for 8×16, 16×32 and 32×64 hashing matrices, respectively.

Table B.1: Accuracy of 1-bit models, by noise level and sampling interval, for a 8×16 hashing matrix.

	Accuracy (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	100.00	100.00	100.00	78.98	71.70	71.70	100.00	91.55	97.15	79.05	71.80	72.45	99.20	74.85	79.20	71.60	71.40	67.90
KNN	71.43	69.65	74.07	71.16	68.63	70.89	76.55	70.80	74.55	72.10	74.60	69.40	74.00	68.20	71.80	67.60	71.85	64.00
Gaussian naïve Bayes	74.39	72.40	74.99	72.13	71.70	70.84	74.40	72.55	72.45	71.65	71.80	71.45	74.40	72.65	73.95	70.80	69.95	64.40
Multinomial naïve Bayes	72.61	71.86	72.99	71.81	71.70	68.73	71.45	71.50	71.80	72.05	71.80	66.10	71.80	71.40	74.65	64.55	63.65	63.35
Categorical naïve Bayes	74.77	71.59	73.58	72.02	71.70	70.89	72.95	71.25	71.60	71.75	70.45	71.70	72.50	70.60	63.65	66.20	63.65	65.75
SVM	100.00	100.00	100.00	90.08	75.53	73.64	100.00	91.25	98.80	89.00	86.75	73.80	99.10	79.65	86.80	75.15	79.65	68.35
Decision tree	100.00	98.01	99.73	90.13	76.17	71.70	97.00	84.70	85.80	81.55	77.15	71.85	96.60	74.00	79.20	72.35	74.95	68.40
Random forest	100.00	98.22	100.00	92.35	84.26	74.50	99.45	88.00	92.00	85.05	79.60	72.65	99.20	74.35	78.55	77.50	77.60	68.50
GBM	100.00	100.00	100.00	99.57	90.89	85.01	99.90	99.15	99.70	96.40	87.00	75.75	99.90	82.75	93.15	82.35	82.30	66.05

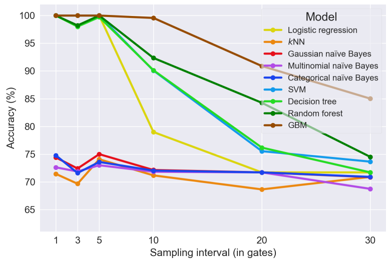
Table B.2: Accuracy of 1-bit models, by noise level and sampling interval, for a 16×32 hashing matrix.

	Accuracy (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	100.00	100.00	99.00	73.60	69.75	60.50	100.00	92.25	88.30	73.15	69.50	60.40	96.75	70.45	75.25	67.15	66.70	61.05
KNN	61.30	60.00	62.20	61.15	61.55	61.10	61.35	59.75	61.05	60.45	61.50	60.80	61.15	60.50	61.25	59.95	59.75	61.05
Gaussian naïve Bayes	70.80	67.20	68.00	60.25	61.10	59.60	70.70	67.85	68.05	65.05	64.40	59.60	70.70	61.35	68.15	63.45	63.65	59.65
Multinomial naïve Bayes	62.05	61.85	64.30	63.50	64.70	59.60	62.05	62.80	64.25	63.60	60.65	60.45	62.10	63.45	64.15	60.50	59.60	59.60
Categorical naïve Bayes	67.30	64.70	67.60	65.90	67.80	62.10	66.80	65.05	67.00	64.90	67.55	61.65	67.50	61.95	67.60	64.50	66.30	61.85
SVM	100.00	100.00	100.00	73.85	70.15	61.65	100.00	92.30	88.50	73.05	68.60	65.70	96.60	70.25	74.80	68.30	65.85	64.65
Decision tree	73.85	71.85	72.55	70.95	69.40	67.15	70.35	70.30	70.55	69.40	68.90	65.65	69.85	67.30	69.30	68.10	68.65	65.30
Random forest	78.80	72.40	72.70	71.95	70.80	67.25	76.85	71.95	73.65	70.90	70.70	66.75	76.55	68.10	72.65	68.40	68.70	64.60
GBM	96.20	91.65	95.15	85.35	78.15	70.75	97.35	92.55	95.70	83.45	77.40	68.30	97.00	74.10	88.30	74.55	74.20	64.10

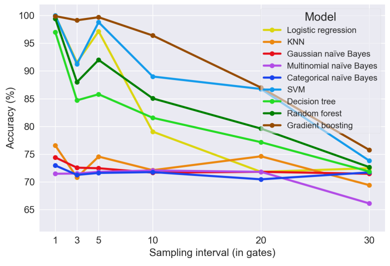
Table B.3: Accuracy of 1-bit models, by noise level and sampling interval, for a 32×64 hashing matrix.

	Accuracy (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	99.95	98.05	87.45	71.85	66.95	60.00	99.25	89.30	86.10	71.20	66.00	60.95	93.55	68.15	75.35	65.70	65.20	60.40
KNN	57.15	57.55	56.20	55.85	57.55	57.55	57.95	56.35	56.75	55.00	56.05	56.55	57.75	56.40	58.10	56.35	57.50	55.65
Gaussian naïve Bayes	65.30	59.55	65.30	58.40	58.95	57.55	65.30	59.80	65.25	59.15	60.50	57.55	65.30	57.55	65.20	58.55	59.40	57.55
Multinomial naïve Bayes	65.00	59.00	64.70	59.25	61.15	57.60	64.80	58.60	64.80	60.10	60.40	57.55	64.85	59.20	64.25	59.05	58.40	57.75
Categorical naïve Bayes	63.20	57.55	62.40	62.05	57.55	57.55	57.55	57.55	63.70	61.10	60.95	57.55	62.75	57.55	57.55	58.30	57.55	57.55
SVM	100.00	100.00	92.80	71.45	65.15	60.30	99.20	89.95	85.95	71.80	65.20	60.15	94.30	66.15	74.65	64.55	65.45	60.15
Decision tree	63.90	64.65	63.00	62.75	63.70	61.00	64.90	64.45	62.55	61.50	61.95	60.55	64.25	62.20	63.15	61.30	61.85	59.45
Random forest	77.15	70.00	72.10	66.20	65.50	62.95	75.90	68.70	70.30	65.50	65.40	62.45	75.10	63.65	69.80	64.95	65.55	60.25
GBM	89.30	83.05	88.10	78.10	69.65	63.70	91.25	83.60	89.25	74.55	68.00	61.10	91.55	70.65	81.20	67.15	67.80	59.00

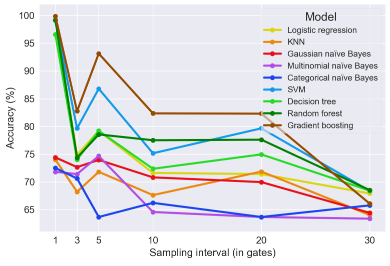
For an easier analysis, figures B.1a to B.1i show their accuracy plotted by sampling interval, for each hashing-matrix size and noise level.



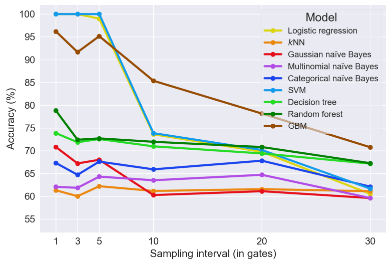
(a) For a 8×16 hashing matrix and no noise.



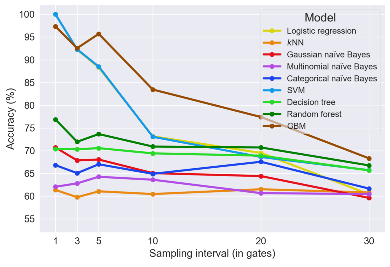
(b) For a 8×16 hashing matrix and noise level of 0.1 pA.



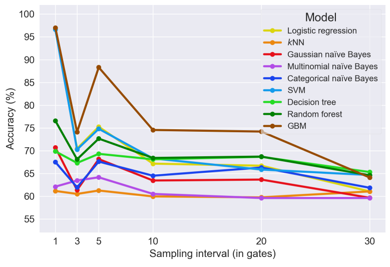
(c) For a 8×16 hashing matrix and noise level of 1 pA.



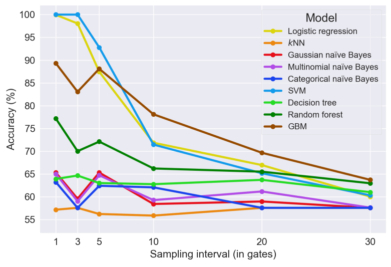
(d) For a 16×32 hashing matrix and no noise.



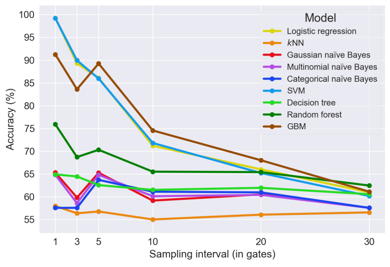
(e) For a 16×32 hashing matrix and noise level of 0.1 pA.



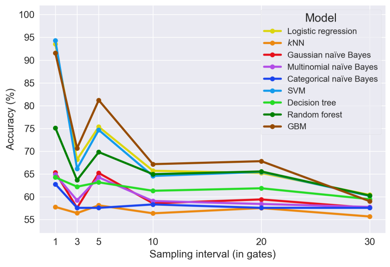
(f) For a 16×32 hashing matrix and noise level of 1 pA.



(g) For a 32×64 hashing matrix and no noise.



(h) For a 32×64 hashing matrix and noise level of 0.1 pA.



(i) For a 32×64 hashing matrix and noise level of 1 pA.

Figure B.1: Accuracy of 1-bit models by sampling interval, for each hashing-matrix size and noise level.

B.2 Full-matrix-model results

For the full-matrix models, the average Hamming loss of the predicted keys, for each combination of noise level and sampling interval, is presented in tables B.4 to B.6, for each hashing-matrix size considered.

Table B.4: Average Hamming loss of the predicted keys of full-matrix models, by noise level and sampling interval, for a 8×16 hashing matrix.

	Average Hamming loss of predicted keys (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	0.00	0.00	0.00	21.87	37.84	34.27	0.00	0.42	0.06	21.29	37.02	34.38	0.00	23.46	18.66	31.26	37.09	38.89
KNN	22.94	27.00	22.70	22.56	25.75	30.32	22.29	27.46	23.09	23.18	23.29	29.12	23.18	30.39	25.40	29.39	25.32	38.56
Gaussian naïve Bayes	0.00	15.53	14.10	26.59	35.79	35.07	0.12	16.80	14.59	25.94	35.08	34.42	1.44	26.32	20.14	30.71	35.26	39.16
Multinomial naïve Bayes	5.32	18.02	17.92	28.73	37.93	34.78	10.42	22.23	19.89	28.22	36.60	34.26	10.91	29.81	24.78	33.81	37.45	39.30
Categorical naïve Bayes	0.00	19.47	10.59	22.81	31.93	34.87	0.00	19.89	11.19	22.15	30.63	34.11	0.50	28.28	18.84	28.81	33.01	39.77
SVM	0.00	0.00	0.00	3.07	4.15	23.30	0.00	0.62	0.01	2.56	4.03	27.32	0.00	14.41	5.73	14.49	10.47	36.19
Decision tree	0.00	0.05	0.00	2.35	14.37	24.86	0.00	1.01	0.31	4.23	16.03	28.52	0.09	13.14	7.83	15.64	16.34	39.12
Random forest	0.00	0.00	0.00	1.84	5.87	15.05	0.00	0.06	0.00	3.20	9.26	18.91	0.06	7.75	3.50	12.15	14.32	37.89
GBM	0.00	0.00	0.00	0.00	1.04	5.08	0.00	0.00	0.00	0.12	5.46	20.13	0.06	1.52	0.41	4.34	10.71	31.28

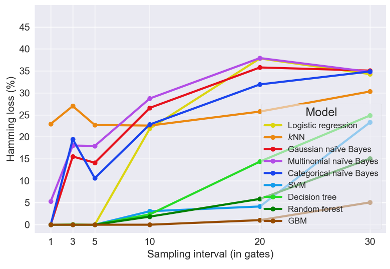
Table B.5: Average Hamming loss of the predicted keys of full-matrix models by noise level and sampling interval, for a 16×32 hashing matrix.

	Average Hamming loss of predicted keys (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	0.00	0.00	0.00	28.59	35.98	46.93	0.00	0.27	2.87	28.51	35.85	46.73	0.00	25.87	20.72	35.39	38.64	46.68
KNN	38.21	40.99	38.24	39.65	39.13	40.77	38.24	37.24	38.25	40.38	39.60	40.72	38.31	41.74	38.36	40.72	40.22	41.29
Gaussian naïve Bayes	0.00	14.17	17.33	32.53	36.59	46.08	3.39	18.07	18.30	32.91	36.52	46.11	6.29	33.29	24.11	36.66	37.37	46.17
Multinomial naïve Bayes	9.83	21.91	23.47	36.05	39.68	46.98	21.27	29.92	25.15	36.57	39.95	47.04	18.04	38.32	29.78	40.67	40.67	46.87
Categorical naïve Bayes	0.00	20.38	8.30	25.85	29.80	44.29	0.00	20.39	8.37	25.83	29.88	44.21	1.60	32.78	19.47	33.91	34.61	45.00
SVM	0.00	0.00	0.00	15.38	22.06	31.37	0.00	0.28	2.84	16.24	23.15	31.85	0.00	27.00	16.48	25.56	26.35	36.97
Decision tree	17.92	23.95	22.38	23.50	25.13	37.88	19.87	26.45	23.68	25.07	26.40	37.43	20.10	30.13	25.84	29.04	28.70	39.14
Random forest	0.00	12.79	2.69	12.15	18.04	31.54	0.00	12.09	2.89	7.37	19.59	34.27	0.03	27.32	11.44	23.35	22.75	38.43
GBM	0.00	0.00	0.00	2.00	11.72	25.67	0.00	0.02	0.01	3.17	18.35	28.96	0.03	1.69	0.56	14.65	20.74	34.94

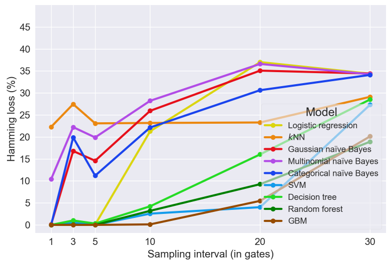
Table B.6: Average Hamming loss of the predicted keys of full-matrix models by noise level and sampling interval, for a 32×64 hashing matrix.

	Average Hamming loss of predicted keys (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	0.00	0.00	0.00	27.03	36.39	44.68	0.00	0.38	4.49	27.05	36.65	45.15	0.00	27.12	19.70	34.76	38.00	45.91
KNN	44.05	46.64	44.29	46.14	45.48	48.00	44.07	46.54	44.28	46.33	46.05	47.97	44.18	47.60	44.43	46.48	46.03	48.17
Gaussian naïve Bayes	1.19	15.01	16.36	32.57	37.68	45.18	9.10	19.70	18.95	33.95	37.80	45.36	11.92	38.34	24.76	37.85	38.40	46.27
Multinomial naïve Bayes	12.43	25.82	27.18	37.16	40.05	45.78	29.71	37.92	30.53	38.65	39.67	45.91	28.39	41.85	31.65	42.12	40.92	46.72
Categorical naïve Bayes	0.27	20.50	10.13	30.12	32.28	43.85	0.30	20.94	10.09	30.15	32.21	43.84	7.55	35.94	22.00	36.45	35.80	45.36
SVM	0.00	0.00	0.00	26.62	34.08	42.85	0.00	0.35	4.26	26.95	34.27	42.60	0.00	26.49	20.06	34.57	34.65	43.89
Decision tree	34.71	35.23	35.35	37.19	37.30	40.34	35.08	36.01	35.96	37.39	37.86	40.47	35.91	37.82	36.37	39.19	39.12	43.47
Random forest	0.01	28.15	23.61	23.80	28.20	37.88	1.89	28.23	21.65	24.93	27.95	35.93	2.71	35.22	21.91	30.20	30.39	42.09
GBM	0.00	0.00	0.00	8.27	15.20	31.34	0.00	0.01	0.00	13.48	29.10	28.56	0.02	0.83	0.50	21.22	32.64	41.12

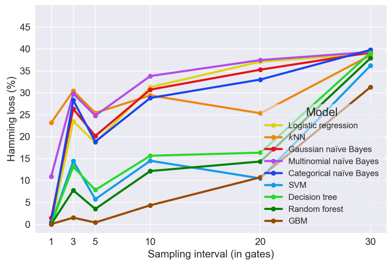
Figures B.2a to B.2i show the average Hamming loss of the predicted keys of these models, plotted by sampling interval, for each hashing-matrix size and noise level.



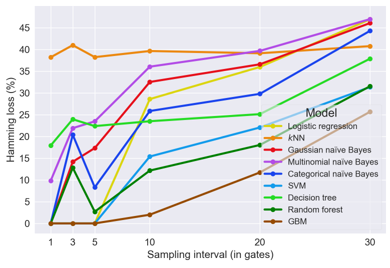
(a) For a 8×16 hashing matrix and no noise.



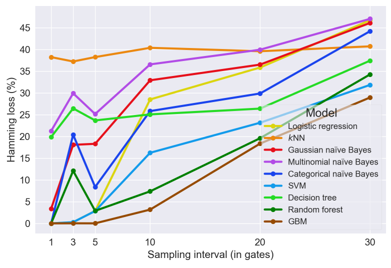
(b) For a 8×16 hashing matrix and noise level of 0.1 pA.



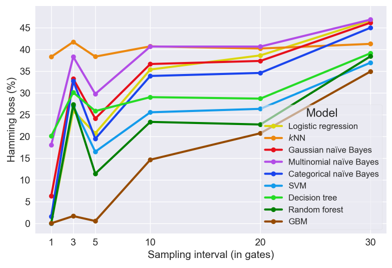
(c) For a 8×16 hashing matrix and noise level of 1 pA.



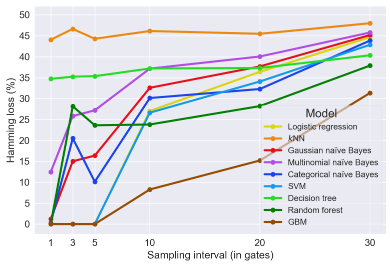
(d) For a 16×32 hashing matrix and no noise.



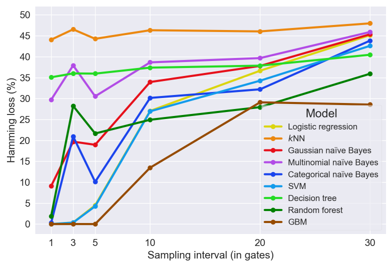
(e) For a 16×32 hashing matrix and noise level of 0.1 pA.



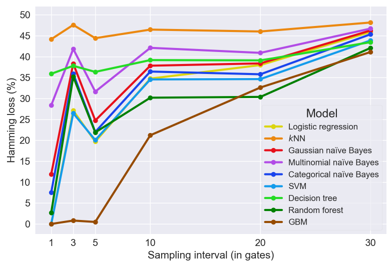
(f) For a 16×32 hashing matrix and noise level of 1 pA.



(g) For a 32×64 hashing matrix and no noise.



(h) For a 32×64 hashing matrix and noise level of 0.1 pA.



(i) For a 32×64 hashing matrix and noise level of 1 pA.

Figure B.2: Average Hamming loss of the predicted keys of full-matrix models by sampling interval, for each hashing-matrix size and noise level.

B.3 Row-model results

For the row models, the average Hamming loss of the predicted keys for each combination of noise level and sampling interval is presented in tables tables B.7 to B.9, for each of the considered hashing-matrix sizes.

Table B.7: Average Hamming loss of the predicted keys of row models, by noise level and sampling interval, for a 8×16 hashing matrix.

	Average Hamming loss of predicted keys (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	0.00	0.00	33.48	44.33	43.22	43.57	0.00	3.45	31.80	42.15	43.75	43.70	0.00	25.95	32.05	43.40	45.10	43.80
KNN	2.41	6.68	6.98	21.29	30.72	30.12	3.35	13.25	11.35	22.25	30.75	28.45	5.10	20.90	16.05	29.50	33.10	37.55
Gaussian naïve Bayes	0.00	15.71	37.90	43.98	45.08	43.42	6.00	20.20	37.10	43.45	44.10	43.70	8.25	27.90	37.80	43.05	43.40	43.70
Multinomial naïve Bayes	9.64	22.29	37.70	44.83	44.53	42.82	22.90	26.00	37.35	43.80	44.10	43.25	23.35	29.55	37.30	43.65	45.25	43.50
Categorical naïve Bayes	0.00	24.05	19.63	25.50	32.98	34.84	0.00	22.30	20.75	28.45	33.00	33.65	0.60	28.00	28.15	37.35	35.35	41.75
SVM	0.00	0.00	2.81	17.92	30.92	30.42	0.00	0.05	2.80	20.75	30.30	29.80	0.00	15.40	13.65	27.00	31.40	40.15
Decision tree	0.00	0.00	2.56	18.22	31.02	29.02	0.00	0.00	5.35	19.15	31.05	28.65	0.00	7.10	13.10	28.50	33.40	38.15
Random forest	0.00	0.00	3.01	18.67	30.87	29.52	0.00	0.00	3.55	18.70	31.35	29.00	0.00	5.15	11.15	26.20	31.85	37.50
GBM	0.00	0.00	3.61	17.42	30.67	28.92	0.00	0.00	3.25	19.00	30.90	28.95	0.00	5.30	11.40	27.25	33.55	39.05

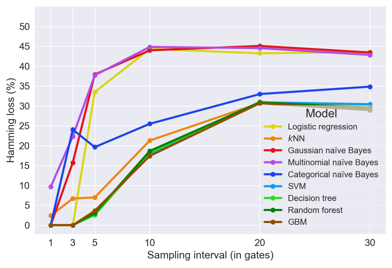
Table B.8: Average Hamming loss of the predicted keys of row models by noise level and sampling interval, for a 16×32 hashing matrix.

	Average Hamming loss of predicted keys (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	0.00	0.00	37.55	42.80	48.80	48.45	0.00	1.90	37.55	43.10	48.60	48.45	0.00	30.10	38.00	43.30	48.75	47.90
KNN	14.90	21.50	25.85	36.20	33.55	42.75	15.15	22.00	25.20	36.35	33.05	39.75	16.30	34.90	28.50	39.85	39.30	43.85
Gaussian naïve Bayes	0.00	13.05	47.70	45.65	49.70	48.35	0.50	14.60	47.80	45.85	49.80	48.00	1.75	30.80	47.75	46.20	49.70	47.95
Multinomial naïve Bayes	4.70	15.55	45.20	46.65	50.80	48.50	16.05	18.65	45.30	46.85	50.55	48.70	15.95	40.35	44.90	46.95	51.65	48.70
Categorical naïve Bayes	0.00	22.10	20.40	33.35	32.70	45.70	0.00	22.25	23.30	35.45	33.40	46.20	0.50	33.95	34.70	42.20	39.90	48.95
SVM	0.00	0.00	10.05	26.40	29.20	42.50	0.00	1.00	10.90	26.80	29.40	41.50	0.00	27.15	19.80	36.10	35.45	43.20
Decision tree	0.00	0.00	6.80	22.65	31.05	42.10	0.00	0.00	6.80	23.55	30.05	40.45	0.05	14.00	15.50	33.50	34.55	41.50
Random forest	0.00	0.00	6.90	23.80	30.85	43.20	0.00	0.00	5.45	23.10	29.30	41.55	0.05	9.00	12.85	31.20	34.15	41.60
GBM	0.00	0.00	6.45	23.75	29.10	40.85	0.00	0.05	5.65	22.80	30.40	41.75	0.05	9.30	13.40	31.80	33.80	43.65

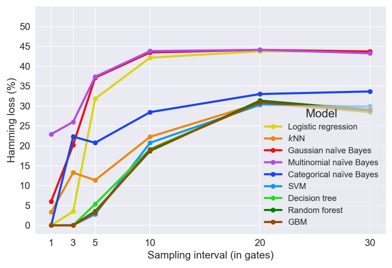
Table B.9: Average Hamming loss of the predicted keys of row models by noise level and sampling interval, for a 32×64 hashing matrix.

	Average Hamming loss of predicted keys (%)																	
Noise level (pA)	0						0.1						1					
Sampling interval (in gates) / Models	1	3	5	10	20	30	1	3	5	10	20	30	1	3	5	10	20	30
Logistic regression	0.00	0.00	37.65	43.11	50.05	47.52	0.00	3.67	38.34	45.54	50.79	47.62	0.00	31.05	38.05	44.00	50.25	48.12
KNN	25.99	30.95	33.13	39.98	38.14	45.83	26.59	32.74	33.28	42.21	38.69	46.63	26.24	39.34	33.68	42.26	39.98	46.92
Gaussian naïve Bayes	0.00	15.97	49.65	50.25	50.25	50.25	0.50	18.95	49.80	50.25	50.25	50.25	2.23	33.68	50.30	50.25	50.25	50.25
Multinomial naïve Bayes	5.70	21.48	42.56	47.52	51.74	48.91	18.95	22.92	42.66	47.42	51.69	50.30	19.59	41.91	44.44	47.02	51.24	50.20
Categorical naïve Bayes	0.00	25.99	21.73	38.10	38.74	47.07	0.00	25.74	23.66	38.39	38.99	47.07	0.30	35.76	48.07	47.57	41.27	48.71
SVM	0.00	0.00	20.19	43.11	37.10	47.02	0.00	3.87	19.99	43.15	36.46	47.02	0.00	30.21	23.86	45.04	37.95	47.52
Decision tree	0.00	0.00	7.39	22.97	31.30	43.01	0.00	0.00	8.09	25.25	32.09	45.24	0.05	9.33	16.37	36.01	38.59	46.92
Random forest	0.00	0.00	7.04	23.66	32.69	46.97	0.00	0.00	7.29	24.65	33.38	46.92	0.05	7.54	15.53	34.42	34.67	47.77
GBM	0.00	0.00	7.49	24.50	32.69	44.35	0.00	0.00	6.85	24.11	32.74	46.68	0.05	9.13	13.59	33.18	36.26	44.44

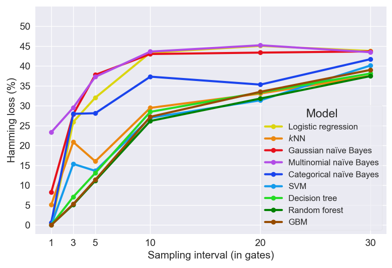
Figures B.3a to B.3i show the average Hamming loss of the predicted keys of the row models, plotted by sampling interval, for each hashing-matrix size and noise level.



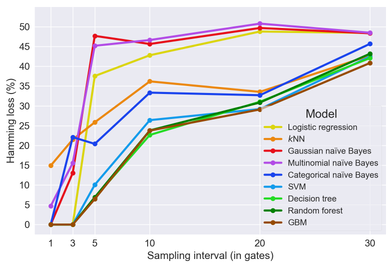
(a) For a 8×16 hashing matrix and no noise.



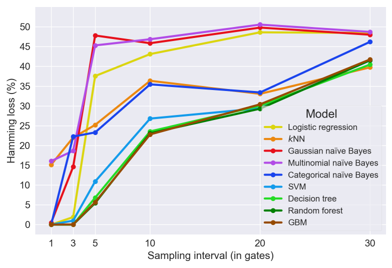
(b) For a 8×16 hashing matrix and noise level of 0.1 pA.



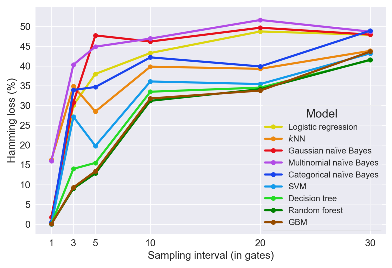
(c) For a 8×16 hashing matrix and noise level of 1 pA.



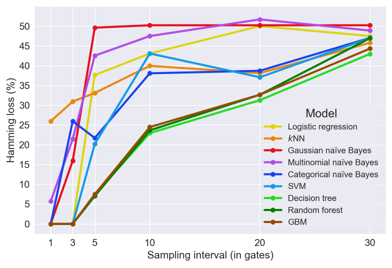
(d) For a 16×32 hashing matrix and no noise.



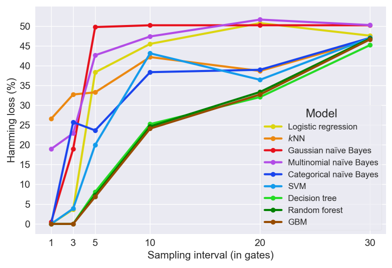
(e) For a 16×32 hashing matrix and noise level of 0.1 pA.



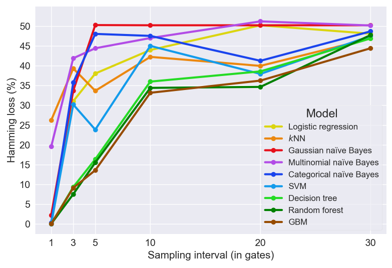
(f) For a 16×32 hashing matrix and noise level of 1 pA.



(g) For a 32×64 hashing matrix and no noise.



(h) For a 32×64 hashing matrix and noise level of 0.1 pA.



(i) For a 32×64 hashing matrix and noise level of 1 pA.

Figure B.3: Average Hamming loss of the predicted keys of row models by sampling interval, for each hashing-matrix size and noise level.