# Applied Cryptography Knowledge Area Version 1.0.0

**Kenneth G. Paterson** | ETH Zürich

EDITOR
**Steve Schneider** | University of Surrey

REVIEWERS
**Dan Bogdanov** | Cybernetica
**Liqun Chen** | University of Surrey
**Keith Martin** | Royal Holloway, University of London
**Nigel Smart** | KU Leuven

## COPYRIGHT

Version 1.0.0 is a stable public release of the Applied Cryptography Knowledge Area.

# CHANGELOG

| Version date | Version number | Changes made |
|---|---|---|
| July 2021 | 1.0 | |

# INTRODUCTION

This document provides a broad introduction to the field of cryptography, focusing on applied aspects of the subject. It complements the Cryptography CyBOK Knowledge Area [1] which focuses on formal aspects of cryptography (including definitions and proofs) and on describing the core cryptographic primitives. That said, formal aspects are highly relevant when considering applied cryptography. As we shall see, they are increasingly important when it comes to providing security assurance for real-world deployments of cryptography.

The overall presentation assumes a basic knowledge of either first-year undergraduate mathematics, or that found in a discrete mathematics course of an undergraduate Computer Science degree. Good cryptography textbooks that cover the required material include [2, 3, 4].

We begin by informally laying out the key themes that we will explore in the remainder of the document.

## Cryptography is a Mongrel

Cryptography draws from a number of fields including mathematics, theoretical computer science and software and hardware engineering. For example, the security of many public key algorithms depends on the hardness of mathematical problems which come from number theory, a venerable branch of mathematics. At the same time, to securely and efficiently implement such algorithms across a variety of computing platforms requires a solid understanding of the engineering aspects. To make these algorithms safely usable by practitioners, one should also draw on usability and Application Programming Interface (API) design. This broad base has several consequences. Firstly, almost no-one understands all aspects of the field perfectly (including the present author). Secondly this creates gaps — between theory and practice, between design and implementation (typically in the form of a cryptographic library, a collection of algorithm and protocol implementations in a specific programming language) and between implementations and their eventual use by potentially non-expert developers. Thirdly, these gaps lead to security vulnerabilities. In fact, it is rare that standardised, widely-deployed cryptographic algorithms directly fail when they are properly used. It is more common that cryptography fails for indirect reasons — through unintentional misuse of a library API by a developer, on account of bad key management, because of improper combination of basic cryptographic algorithms in a more complex system, or due to some form of side-channel leakage. All of these topics will be discussed in more detail.

## Cryptography $\neq$ Encryption

In the popular imagination cryptography equates to encryption; a cryptographic mechanism providing confidentiality services. In reality, cryptography goes far beyond this to provide an underpinning technology for building security services more broadly. Thus, secure communications protocols like Transport Layer Security (TLS) rely on both encryption mechanisms (Authenticated Encryption, AE) and integrity mechanisms (e.g. digital signature schemes) to achieve their security goals. In fact, in its most recent incarnation (version 1.3), TLS relies exclusively on Diffie-Hellman key exchange to establish the keying material that it consumes, whereas earlier versions allowed the use of public key encryption for this task. We will discuss TLS more extensively in Section 5; here the point is that, already in the literally classic application of cryptography, encryption is only one of many techniques used.

Moreover, since the boom in public research in cryptography starting in the late 1970s, researchers have been incredibly fecund in inventing new types of cryptography to solve seemingly impossible tasks. Whilst many of these new cryptographic gadgets were initially of purely theoretical interest, the combination of Moore's law and the growth of technologies such as cloud computing has made some of them increasingly important in practice. Researchers have developed some of these primitives to the point where they are efficient enough to be used in large-scale applications. Some examples include the use of zero-knowledge proofs in anonymous cryptocurrencies, the use of Multi-Party Computation (MPC) techniques to enable computations on sensitive data in environments where parties are mutually untrusting, and the (to date, limited) use of Fully Homomorphic Encryption (FHE) for privacy-preserving machine learning.

## Cryptography is Both Magical and Not Magical

Cryptography can seem magical in what it can achieve. Consider the millionaire's problem: two millionaires want to find out who is richer, without either telling the other how much they are worth. This seems impossible, but it can be done in a reasonably efficient manner and under mild assumptions. But there is no such thing as cryptographic "fairy dust" that can be sprinkled on a bad (i.e. insecure) system to make it good (i.e. secure). Rather, at the outset of a system design activity, cryptography should be thought of as being something that will work in concert with other security building blocks to build a secure system (or more pessimistically, a system in which certain risks have been mitigated). In this sense, cryptography makes systems stronger by making certain attack vectors infeasible or just uneconomic to attack. Consider again the secure communications protocol TLS. When configured properly, TLS offers end-to-end security for pairs of communicating devices, providing strong assurances concerning the confidentiality and integrity of messages (and more). However, it says nothing about the security of the endpoints where the messages are generated and stored. Moreover, TLS does not prevent traffic analysis attacks, based on analysing the number, size and direction of flow of TLS-encrypted messages. So the use of cryptography here reduces the "attack surface" of the communication system, but does not eliminate all possible attacks on the system, nor does it aim to do so.

Developing the systemic view further, it is unfortunate that cryptography is generally brittle and can fail spectacularly rather than gracefully. This can be because of a breakthrough in cryptanalysis, in the sense of breaking one of the core cryptographic algorithms in use. For example, there could be an unexpected public advance in algorithms for integer factorisation that renders our current choices of key-sizes for certain algorithms totally insecure. This seems unlikely — the last significant advance at an algorithmic level here was in the early 1990s with the invention of the Number Field Sieve. So more likely this is because the cryptography is provided in a way that makes it easy for non-experts to make mistakes, or the realisation of a new attack vector enabling an attacker to bypass the cryptographic mechanism in use.

It is also an unfortunate fact that, in general, cryptography is non-composable, in the sense that a system composed of cryptographic components that are individually secure (according to some suitable formal definitions for each component) might itself fail to be secure in the intended sense. A simple example arises in the context of so-called *generic composition* of symmetric encryption and MAC algorithms to build an overall encryption scheme that offers both confidentiality and integrity. Here, the "E&M" scheme obtained by encrypting the plaintext and, in parallel, applying a MAC to the plaintext, fails to offer even a basic level of confidentiality. This is because the MAC algorithm, being deterministic, will leak plaintext equality across multiple encryptions. This example, while simple, is not artificial: the SSH protocol historically used such an E&M scheme and only avoided the security failure due to the inclusion of a per-message sequence number as part of the plaintext (this sequence number was also needed to achieve other security properties of the SSH secure channel). This example generalises, in the sense that even small and seemingly trivial details can have a large effect on security: in cryptography, every bit matters.

In view of the above observations, applied cryptography is properly concerned with a broader sweep of topics than just the low-level cryptographic algorithms. Of course these are still crucial and we will cover them briefly. However, applied cryptography is also about the integration of cryptography into systems and development processes, the thorny topic of key management and even the interaction of cryptography with social processes, practices and relations. We will touch on all of these aspects.

## Cryptography is Political

Like many other technologies, cryptography can be used for good or ill. It is used by human rights campaigners to securely organise their protests using messaging apps like Telegram and Signal [5]. It is used by individuals who wish to maintain their privacy against the incursions of tech companies. It enables whistle-blowers to securely communicate with journalists when disclosing documents establishing company or governmental wrong-doing (see Privacy & Online Rights CyBOK Knowledge Area [6]). But it can also be used by terrorists to plan attacks or by child-abusers to share illegal content. Meanwhile cryptocurrencies can be used by drug dealers to launder money [7] and as a vehicle for extracting ransom payments.[1]

These examples are chosen to highlight that cryptography, historically the preserve of governments and their militaries, is now in everybody's hands — or more accurately, on everybody's phone. This is despite intensive, expensive efforts over decades on the part of governments to regulate the use of cryptography and the distribution of cryptographic technology through export controls. Indeed, such laws continue to exist, and violations of them can produce severe negative consequences so practitioners should be cautious to research applicable regulation (see Law & Regulation CyBOK Knowledge Area [9] for further discussion of this topic).

But the cryptographic genie has escaped the bottle and is not going back in. Indeed, cryptographic software of reasonable quality is now so widespread that attempts to prevent its use or to introduce government-mandated back-doors are rendered irrelevant for anyone with a modicum of skill. This is to say nothing as to whether it is even possible to securely engineer cryptographic systems that support exceptional access for a limited set of authorised parties, something which experts doubt, see for example [10]. Broadly, these efforts at control and the reaction to them by individual researchers, as well as companies, are colloquially known as The Crypto Wars. Sometimes, these are enumerated, though it is arguable that the First Crypto War never ended, but simply took on another, less-intense, less-visible form, as became apparent from the Snowden revelations [11].

[1]On the other hand, a 2019 RAND report [8] concluded there is little evidence for use of cryptocurrencies by terrorist groups.

## The Cryptographic Triumvirate

A helpful classification of cryptographic applications arises from considering what is happening to the data. The classical cryptographic applications relate to *data in transit*, i.e. secure communications. Cryptography can be applied to build secure data storage systems, in which case we talk about *data at rest*. In fact these two application domains are quite close in terms of the techniques they use. This is because, to a first-order approximation, one can regard a secure storage system as a kind of communications channel in time. Finally, in the era of cloud computing, outsourced storage and privacy-preserving computation, we have seen the emergence of cryptography for *data under computation*. This refers to a broad set of techniques enabling computations to be done on data that is encrypted — imagine outsourcing an entire database to an untrusted server in such a way that database operations (insert, delete, search queries, etc) can still be carried out without leaking anything about those operations — or the data itself — to the server. Keeping in mind this triumvirate — data in transit, data at rest, data under computation — can be useful when understanding what to expect in terms of the security, performance and maturity of systems using cryptography. In short, systems providing security for data in transit and data at rest are more mature, are performant at scale and tend to be standardised. By contrast, systems providing security for data under computation are largely in an emergent phase.

This classification focuses on data and therefore fails to capture some important applications of cryptography such as user authentication[2] and attestation.[3]

## Organisation

Having laid out the landscape of Applied Cryptography, we now turn to a more detailed consideration of sub-topics. The next section is concerned with cryptographic algorithms and schemes — the building blocks of cryptography. It also discusses protocols, which typically combine multiple algorithms into a more complex system. In Section 2 we discuss implementation aspects of cryptography, addressing what happens when we try to turn a mathematical description of a cryptographic algorithm into running code. Cryptography simply translates the problem of securing data into that of securing and managing cryptographic keys, following Wheeler's aphorism that every problem in computer science can be solved by another level of indirection. We address the topic of key management in Section 3. Section 4 covers a selection of issues that may arise for non-expert consumers of cryptography, while Section 5 discusses a few core cryptographic applications as a means of showing how the different cryptographic threads come together in specific cases. Finally, Section 6 looks to the future of applied cryptography and conveys closing thoughts.

[2]Here, for example, FIDO is developing open specifications of interfaces for authenticating users to web-based applications and services using public key cryptography.

[3]This concept refers to methods by which a hardware platform can provide security guarantees to third parties about how it will execute code. It is addressed in the Hardware Security CyBOK Knowledge Area [12].

# 1 ALGORITHMS, SCHEMES AND PROTOCOLS

[2, 3, 4]

## 1.1 Basic Concepts

In this subsection, we provide a brief summary of some basic concepts in cryptography. A more detailed and formal introduction to this material can be found in Cryptography CyBOK Knowledge Area [1].

Cryptographic algorithms are at the core of cryptography. There are many different classes of algorithm and many examples within each class. Moreover, it is common to group algorithms together to form cryptographic *primitives* or *schemes*. For example, a *Public Key Encryption (PKE) scheme* consists of a collection of three algorithms: a *key generation* algorithm, an *encryption* algorithm and a corresponding *decryption* algorithm.

Unfortunately, there is no general agreement on the terminology and the meanings of the terms *algorithm*, *scheme*, *primitive* and even *protocol* overlap and are sometimes used interchangeably. We will reserve the term *algorithm* for an individual algorithm (in the computer science sense — a well-defined procedure with specific inputs and outputs, possibly randomised). We will use *scheme* to refer to a collection of algorithms providing some functionality (e.g. as above, a PKE scheme) and *protocol* for interactive systems in which two or more parties exchange messages.[4] Such protocols are usually built by combining different cryptographic schemes, themselves composed of multiple algorithms.

Most cryptographic algorithms are *keyed* (the main exception are hash functions). This means that they have a special input, called a key, which controls the operation of the algorithm. The manner in which the keying is performed leads to the fundamental distinction between symmetric and asymmetric schemes. In a symmetric scheme the same key is used for two operations (e.g. encryption and decryption) and the confidentiality of this key is paramount for the security of the data which it protects. In an asymmetric scheme, different keys are used for different purposes (e.g. in a PKE scheme, a public key is used for encryption and a corresponding private key is used for decryption, with the *key pair* of public and private keys being output by the PKE scheme's key generation algorithm). The usual requirement is that the private key remains confidential to the party who runs the key generation algorithm, while the public key (as the name suggests) can be made public and widely distributed.

We now turn to discussion of the most important (from the perspective of applied cryptography) cryptographic primitives and schemes. Our treatment is necessarily informal and incomplete. Any good textbook will provide missing details. Martin's book [3] provides an accessible and mostly non-mathematical treatment of cryptography. Smart's book [4] is aimed at Computer Science undergraduates with some background in mathematics. The text by Boneh and Shoup [2] is more advanced and targets graduate students.

[4]Alternatively, a protocol is a distributed algorithm.

## 1.2 Hash functions

A hash function is usually an unkeyed function $H$ which takes as input bit-strings of variable length and produces short outputs of some fixed length, $n$ bits say.

A crucial security property is that it should be hard to find collisions for a hash function, that is, pairs of inputs $(m_0, m_1)$ resulting in the same hash value, i.e. such that $H(m_0) = H(m_1)$ (such collisions must exist because the output space is much smaller than the input space). If the output size is $n$ bits, then there is a generic attack based on the birthday paradox that will find collisions with effort about $2^{n/2}$ hash function evaluations.[5]

Other important security properties are pre-image resistance and second pre-image resistance. Informally, pre-image resistance says that it is hard, given an output from a hash function $h$, to find an input $m$ such that $H(m) = h$. Second pre-image resistance says that, given an $m$, it is difficult to find $m' \neq m$ such that $H(m) = H(m')$.

Hash functions are often modelled as random functions in formal security analyses, leading to the Random Oracle Model. Of course, a given hash function is a *fixed* function and not a random one, so this is just a heuristic, albeit a very useful one when performing formal security analyses of cryptographic schemes making use of hash functions.

SHA-1 with $n = 160$ is a widely-used hash function. However, collisions for SHA-1 can be found using much less than $2^{80}$ effort, so it is now considered unsuitable for applications requiring collision-resistance. Other common designs include SHA-256 (with $n = 256$ and still considered very secure) and SHA-3 (with variable length output and based on different design principles from the earlier SHA families). The SHA families are defined in a series of NIST standards [13, 14].

## 1.3 Block ciphers

A block cipher is a function taking as input a symmetric key $K$ of $k$ bits and a plaintext $P$ with $n$ bits and producing as output a ciphertext $C$ also of $n$ bits. For each choice of $K$, the resulting function, often written $E_K(\cdot)$, is a permutation mapping $n$-bit strings to $n$-bit strings. Since $E_K(\cdot)$ is a permutation, it has an inverse, which we denote $D_K(\cdot)$. We require that both $E_K(\cdot)$ and $D_K(\cdot)$ be fast to compute.

Many security properties can be defined for block ciphers, but the most useful one for formal security analysis demands that the block cipher be a Pseudo-Random Permutation (PRP). Informally this means, if $K$ is chosen uniformly at random, then no efficient adversary can tell the difference between outputs of the block cipher and the outputs of a permutation selected uniformly at random from the set of all permutations of $n$ bits.

It is notable that block cipher designers do not typically target such a goal, but rather resistance to a range of standard attacks. One such attack is exhaustive key search: given a few known plaintext/ciphertext pairs for the given key, try each possible $K$ to test if it correctly maps the plaintexts to the ciphertexts. This *generic* attack means that a block cipher's key length $k$ must be big enough to make the attack infeasible. The Data Encryption Standard (DES) had

[5]The birthday paradox is a generalisation of the initially surprising observation that in a group of 23 randomly selected people there is a 50-50 chance of two people sharing a birthday; more generally, if we make $\sqrt{N}$ selections uniformly at random from a set of $N$ objects, then there is a constant probability of two of the selections being the same.

$k = 56$ which was considered by experts already too short when the algorithm was introduced by the US government in the mid 1970s.

The Advanced Encryption Standard (AES) [15] is now the most-widely used block cipher. The AES was the result of a design competition run by the US government agency NIST. It has a 128-bit block ($n = 128$) and its key-length $k$ is either 128, 192, or 256, precluding exhaustive key search. Fast implementation of AES is supported by hardware instructions on many Central Processing Unit (CPU) models. Fast and secure implementation of AES is challenging in environments where an attacker may share memory resources with the victim, for example a cache. Still, with its widespread support and lack of known security vulnerabilities, it is rarely the case that any block cipher other than AES is needed. One exception to this rule is constrained computing environments.

Except in very limited circumstances, block ciphers should not be used directly for encrypting data. Rather, they are used in *modes of operation* [16]. Modes are discussed further below under Authenticated Encryption schemes.

## 1.4 Stream ciphers

Stream ciphers are algorithms that can encrypt a stream of bits (as opposed to a block of $n$ bits in the case of block ciphers) under the control of a $k$-bit key $K$. Most stream ciphers use the key to generate a key-stream and then combine it in a bit-by-bit fashion with the plaintext using an XOR operation, to produce the ciphertext stream.

Keystream reuse is fatal to security, since the XOR of two ciphertexts created using the same keystream reveals the XOR of the plaintexts, from which recovering the individual plaintexts becomes possible given enough plaintext redundancy [17]. To avoid this, stream ciphers usually employ an Initialisation Vector (IV) along with the key; the idea is that each choice of IV should produce an independent keystream. IVs need not be kept secret and so can be sent along with the ciphertext or derived by sender and receiver from context.

The main security requirement for a stream cipher is that, for a random choice of key $K$, and each choice of IV, it should be difficult for an adversary to distinguish the resulting keystream from a truly random bit-string of the same length.

It is easy to build a stream cipher from a block cipher by using a *mode of operation*; these are discussed in Section 1.6.4. Dedicated stream ciphers suitable for implementation in hardware have traditionally relied on combining simpler but insecure components such as Linear Feedback Shift Registers. The A5/1 and A5/2 stream ciphers once widely used in mobile telecommunications systems are of this type. The RC4 stream cipher was well-suited for implementation in software and has a very simple description making it attractive to developers. RC4 became widely used in IEEE wireless communications systems (WEP, WPA) and in Secure Socket Layer/Transport Layer Security (SSL/TLS). It is now considered obsolete because of a variety of security vulnerabilities that it presents in these applications.

## 1.5 Message Authentication Code (MAC) schemes

MAC schemes are used to provide authentication and integrity services. They are keyed. A MAC scheme consists of three algorithms. The first is called `KeyGen` for key generation. It is randomised and usually consists of selecting a key $K$ at random from the set of bit-strings of some fixed length $k$. The second algorithm is called `Tag`. Given as input $K$ and a message $m$ encoded as a bit-string, `Tag` produces a MAC tag $\tau$, usually a short string of some fixed length $t$. The third algorithm is called `Verify`. This algorithm is used to verify the validity of a message/MAC tag combination $(m, \tau)$ under the key $K$. So `Verify` takes as input triples $(K, m, \tau)$ and produces a binary output, with "1" indicating validity of the input triple.

The main security property required of a MAC scheme is that it should be hard for an adversary to come up with a new pair $(m, \tau)$ which `Verify` accepts with key $K$, even when the adversary has already seen many pairs $(m_1, \tau_1), (m_2, \tau_2), \ldots$ produced by `Tag` with the same key $K$ for messages of its choice $m_1, m_2, \ldots$. The formal security notion is called Strong Unforgeability under Chosen Message Attack (SUF-CMA for short).

SUF-CMA MAC schemes can be used to provide data origin authentication and data integrity services. Suppose two parties Alice and Bob hold a shared key $K$; then no party other than those two can come up with a correct MAC tag $\tau$ for a message $m$. So if Alice attaches MAC tags $\tau$ to her messages before sending them to Bob, then Bob, after running `Verify` and checking that it accepts, can be sure the messages only came from Alice (or maybe himself, depending on how restrictive he is in using the key) and were not modified by an attacker.

MAC schemes can be built from hash functions and block ciphers. HMAC [18] is a popular MAC scheme which, roughly speaking, implements its `Tag` algorithm by making two passes of a hash function applied to the message $m$ prefixed with the key $K$. The security analysis of HMAC requires quite complex assumptions on the underlying hash function [2, Section 8.7]. MAC schemes can also be built from so-called *universal hash functions* in combination with a block cipher. Security then relies only on the assumption that the block cipher is a PRP. Since universal hash functions can be very fast, this can lead to very efficient MACs. A widely used MAC of this type is GMAC, though it is usually used only as part of a more complex algorithm called AES-GCM that is discussed below.

## 1.6 Authenticated Encryption (AE) schemes

An Authenticated Encryption (AE) scheme [19] is a symmetric scheme which transforms plaintexts into ciphertexts and which simultaneously offers confidentiality and integrity properties for the plaintext data. After a long period of debate, AE has emerged as a powerful and broadly applicable primitive for performing symmetric encryption. In most cases where symmetric encryption is needed, AE is the right tool for the job.

An AE scheme consists of three algorithms: `KeyGen`, `Enc` and `Dec`. The first of these is responsible for key generation. It is randomised and usually consists of selecting a key $K$ at random from the set of bit-strings of some fixed length $k$. Algorithm `Enc` performs encryption. It takes as input a key $K$ and a plaintext $M$ to be encrypted. Practical AE schemes allow $M$ to be a bit-string of variable length. In the *nonce-based* setting, `Enc` has an additional input called the *nonce*, denoted $N$, and usually selected from a set of bit-strings of some fixed size $n$. In this setting, `Enc` is deterministic (i.e. it needs no internal randomness). In the *randomised* setting, `Enc` is a randomised algorithm and does not take a nonce input. The third algorithm in an AE scheme is the decryption algorithm, `Dec`. It takes as input a key $K$, a ciphertext string $C$ and, in the nonce-based setting, a nonce $N$. It returns either a plaintext $M$ or an error message indicating that decryption failed. Correctness of a nonce-based AE scheme demands that, for all keys $K$, all plaintexts $M$ and all nonces $N$, if running Enc on input $(K, M, N)$ results in ciphertext $C$, then running Dec on input $(K, C, N)$ results in plaintext $M$. Informally, correctness means that, for a given key, decryption "undoes" encryption.

### 1.6.1 AE Security

Security for nonce-based AE is defined in terms of the combination of a confidentiality property and an integrity property.

Confidentiality for AE says, roughly, that an adversary learns nothing that it does not already know from encryptions of messages. Slightly more formally, we consider an adversary that has access to a *left-or-right (LoR) encryption oracle*. This oracle takes as input a pair of equal-length plaintexts $(M_0, M_1)$ and a nonce $N$ selected by the adversary; internally the oracle maintains a randomly generated key $K$ and a randomly sampled bit $b$. It selects message $M_b$, encrypts it using Enc on input $(K, M_b, N)$ and returns the resulting ciphertext $C$ to the adversary. The adversary's task is to make an estimate of the bit $b$, given repeated access to the oracle while, in tandem, performing any other computations it likes. The adversary is considered successful if, at the end of its attack, it outputs a bit $b'$ such that $b' = b$. An AE scheme is said to be IND-CPA secure ("indistinguishability under chosen plaintext attack") if no adversary, consuming reasonable resources (quantified in terms of the computational resources it uses, the number of queries it makes and sometimes the bit-length of those queries) is able to succeed with probability significantly greater than $1/2$. An adversary can always just make a complete guess $b'$ for the bit $b$ and will succeed half of the time; hence we penalise the adversary by demanding it do significantly better than this trivial guessing attack. The intuition behind IND-CPA security is that an adversary, even with perfect control over which pairs of messages get encrypted, cannot tell from the ciphertext which one of the pair — the left message or the right message — gets encrypted each time. So the ciphertexts do not leak anything about the messages, except perhaps their lengths.[6]

The integrity property says, roughly, that an adversary cannot create new ciphertexts that decrypt to plaintexts, instead of producing decryption failures. Slightly more formally, we give an adversary an encryption oracle; this oracle internally maintains a randomly generated key $K$ and on input $(M, N)$ from the adversary, runs Enc on input $(K, M, N)$ and returns the resulting ciphertext $C$ to the adversary. We also give the adversary a decryption oracle, to which it can send inputs $(C, N)$. In response to such inputs, the oracle runs Dec on input $(K, C, N)$ (for the same key $K$ used in the encryption oracle) and gives the resulting output to the adversary — this output could be a message or an error message. The adversary against integrity is considered to be successful if it at some point sends an input $(C, N)$ to its decryption oracle which results in an output that is a plaintext $M$ and not an error message. Here, we require that $(C, N)$ be such that $C$ is *not* a ciphertext output by the encryption oracle when it was given some input $(M, N)$ during the attack — otherwise the adversary can win trivially. Under this restriction, for the adversary to win, the pair $(C, N)$ must be something new that the adversary could not have trivially obtained from its encryption oracle. In this sense, $C$ is a ciphertext forgery for some valid plaintext, which the adversary may not even know before it sends $(C, N)$ for decryption. An AE scheme is said to be INT-CTXT secure (it has "integrity of ciphertexts")

[6] But note that hiding the length of data can be a more complicated task than hiding the data itself. Moreover, just knowing the length of data can lead to significant attacks. Such attacks, more broadly covered under side-channel attacks, are discussed in Section 2.3.

if no adversary, consuming reasonable resources (quantified as before) is able to succeed with probability significantly greater than $0$.

Similar security notions can be developed for the randomised setting.

An AE scheme is said to be secure (or AE secure) if it is both IND-CPA and INT-CTXT secure. This combination of security properties is very strong. It implies, for example, IND-CCA security, a traditional security notion which extends IND-CPA security by also equipping the adversary with a decryption capability, capturing the capabilities that a practical attacker may have. It also implies a weaker "integrity of plaintexts" notion, which roughly states that it should be hard for the adversary to create a ciphertext that encrypts a new plaintext.

### 1.6.2 Nonces in AE

It is a requirement in the IND-CPA security definition that the nonces used by the adversary be *unique* across all calls to its LoR encryption oracle. Such an adversary is called a *nonce respecting adversary*. In practice, it is usually the responsibility of the application using an AE scheme to ensure that this condition is met across all invocations of the Enc algorithm for a given key $K$. Note that the nonces do not need to be random. Indeed choosing them randomly may result in nonce collisions, depending on the quality of the random bit source used, the size of the nonce space and the number of encryptions performed. For example, the nonces could be invoked using a stateful counter. The core motivation behind the nonce-based setting for AE is that it is easier for a cryptographic implementation to maintain state across all uses of a single key than it is to securely generate the random bits needed to ensure security in the randomised setting. This is debatable and nonce repetitions have been observed in practice [20]. For some AE schemes such as the widely deployed AES-GCM scheme, the security consequences of accidental nonce repetition are severe, e.g. total loss of integrity and/or partial loss of confidentiality. For this reason, misuse-resistant AE schemes have been developed. These are designed to fail more gracefully under nonce repetitions, revealing less information in this situation than a standard AE scheme might. They are generally more computationally expensive than standard AE schemes. AES-GCM-SIV [21] is an example of such a scheme.

### 1.6.3 AE Variants

Many variants of the basic AE formulation and corresponding security notions have been developed. As an important example, Authenticated Encryption with Associated Data (AEAD) [22] refers to an AE extension in which an additional data field, the Associated Data (AD), is cryptographically bound to the ciphertext and is integrity protected (but not made confidential). This reflects common use cases. For example, we have a packet header that we wish to integrity protect but which is needed in the clear to deliver data, and a packet body that we wish to both integrity protect and make confidential. Even the basic AE security notion can be strengthened by requiring that ciphertexts be indistinguishable from random bits or by considering security in the multi-user setting, where the adversary interacts with multiple AE instantiations under different, random keys and tries to break any one of them. The latter notion is important when considering large-scale deployments of AE schemes. The two separate notions, IND-CPA and INT-CTXT, can also be combined into a single notion [23].

### 1.6.4 Constructing AE Schemes

Secure AE (and AEAD) schemes can be constructed *generically* from simpler encryption schemes offering only IND-CPA security and SUF-CMA secure MAC schemes. There are three basic approaches: Encrypt-then-MAC (EtM), Encrypt-and-MAC (E&M) and MAC-then-Encrypt (MtE). Of these, the security of EtM is the easiest to analyse and provides the most robust combination, because it runs into fewest problems in implementations. Both MtE and E&M have been heavily used in widely-deployed secure communications protocols such as SSL/TLS and Secure Shell (SSH) with occasionally disastrous consequences [24, 25, 26]. Broad discussions of generic composition can be found in [27] (in the randomised setting) and [28] (more generally).

This generic approach then leaves the question of how to obtain an encryption scheme offering IND-CPA security. This is easily achieved by using a block cipher in a suitable mode of operation [16], for example, counter (CTR) mode or CBC mode. Such a mode takes a block cipher and turns it into a more general encryption algorithm capable of encrypting messages of variable length, whereas a block cipher can only encrypt messages of length $n$ bits for some fixed $n$. The IND-CPA security of the mode can then be proved based on the assumption that the used block cipher is a PRP. Indeed, the nonce-based AEAD scheme AES-GCM [16] can be seen as resulting from a generic EtM construction applied using AES in a specific nonce-based version of CTR mode and an SUF-CMA MAC constructed from a universal hash function based on finite field arithmetic. AES-GCM is currently used in about 90% of all TLS connections on the web. It has excellent performance on commodity CPUs) from Intel and AMD because of their hardware support for the AES operations and for the finite field operations required by the MAC. A second popular AEAD scheme, ChaCha20-Poly1305 [29], arises in a similar way from different underlying building blocks. The CAESAR competition[7] was a multi-year effort to produce a portfolio of AEAD schemes for three different use cases: lightweight applications, high-performance applications and defence in depth (essentially, misuse-resistant AE).

## 1.7 Public Key Encryption Schemes and Key Encapsulation Mechanisms

A Public Key Encryption (PKE) scheme consists of three algorithms: KeyGen, Enc and Dec. The first of these is responsible for key generation. It is randomised and outputs key pairs $(sk, pk)$ where $sk$ denotes a private key (often called the secret key) and $pk$ denotes a public key. The algorithm Enc performs encryption. It takes as input the public key $pk$ and a plaintext $M$ to be encrypted and returns a ciphertext $C$. In order to attain desirable security notions (introduced shortly), Enc is usually randomised. The plaintext $M$ comes from some set of possible plaintexts that the scheme can handle. Usually this set is limited and dictated by the mathematics from which the PKE scheme is constructed. This limitation is circumvented using *hybrid encryption*, which combines PKE and symmetric encryption to allow a more flexible set of plaintexts. The third algorithm in a PKE scheme is the decryption algorithm, Dec. It takes as input the private key $sk$ and a ciphertext $C$. It returns either a plaintext $M$ or an error message indicating that decryption failed. Correctness of a PKE scheme requires that, for all key pairs $(sk, pk)$ and all plaintexts $M$, if running Enc on input $(pk, M)$ results in ciphertext $C$, then running Dec on input $(sk, C)$ results in plaintext $M$. Informally, correctness means that, for a given key, decryption “undoes” encryption. Notice here the fundamental asymmetry in the use of keys in a PKE scheme: $pk$ is used during encryption and $sk$ during decryption.

[7] See https://competitions.cr.yp.to/caesar-submissions.html.

### 1.7.1 PKE Security

There are many flavours of security for PKE. We focus on just one, which is sufficient for many applications, and provide a brief discussion of some others.

Recall the definition of IND-CPA and IND-CCA security for AE schemes from Section 1.6. Analogous notions can be defined for PKE. In the IND-CPA setting for PKE, we generate a key pair $(sk, pk)$ by running KeyGen and give the public key $pk$ to an adversary (since public keys are meant to be public!). The adversary then has access to an LoR encryption oracle which, on input a pair of equal-length messages $(M_0, M_1)$, performs encryption of $M_b$ under the public key $pk$, i.e. runs the randomised algorithm Enc on input $(pk, M_b)$, to get a ciphertext $C$ which is then returned to the adversary. The adversary's task is to make an estimate of the bit $b$, given repeated access to the oracle while, in tandem, performing any other computations it likes. The adversary is considered successful if, at the end of its attack, it outputs a bit $b'$ such that $b' = b$. A PKE scheme is said to be IND-CPA secure ("indistinguishability under chosen plaintext attack") if no adversary, consuming reasonable resources (quantified in terms of the computational resources it uses and the number of queries it makes) is able to succeed with probability significantly greater than $1/2$. The intuition behind IND-CPA security for PKE is the same as that for AE: even with perfect control over which pairs of messages get encrypted, an adversary cannot tell from the ciphertext which one of the pairs is encrypted each time.

Note that in order to be IND-CPA secure, a PKE scheme must have a randomised encryption algorithm (if Enc was deterministic, then an adversary that first makes an encryption query on the pair $(M_0, M_1)$ with $M_0 \neq M_1$ and then an encryption query on $(M_0, M_0)$ could easily break the IND-CPA notion). If a PKE scheme is IND-CPA secure, then it must be computationally difficult to recover $pk$ from $sk$, since, if this were possible, then an adversary could first recover $sk$ and then decrypt one of the returned ciphertexts $C$ and thereby find the bit $b$.

IND-CCA security for PKE is defined by extending the IND-CPA notion to also equip the adversary with a decryption oracle. The adversary can submit (almost) arbitrary bit-strings to this oracle. The oracle responds by running the decryption algorithm and returning the resulting plaintext or error message to the adversary. To prevent trivial wins for the adversary and therefore avoid a vacuous security definition, we have to restrict the adversary to not make decryption oracle queries for any of the outputs obtained from its encryption oracle queries.

We do not generally consider integrity notions for PKE schemes. This is because, given the public key $pk$, an adversary can easily create ciphertexts of its own, so no simple concept of "ciphertext integrity" would make sense for PKE. Integrity in the public key setting, if required, usually comes from the application of digital signatures, as discussed in Section 1.9. Digital signatures and PKE can be combined in a cryptographic primitive called *signcryption*. This can be a useful primitive in some use-cases, e.g. secure messaging (see Section 5.2).

In some applications, such as anonymous communications or anonymous cryptocurrencies, anonymity of PKE plays a role. Roughly speaking, this says that a PKE ciphertext should not leak anything about the public key $pk$ that was used to create it. This is an orthogonal property to IND-CPA/IND-CCA security. A related concept is robustness for PKE, which informally says that a ciphertext generated under one public key $pk$ should not decrypt correctly under the private key $sk'$ corresponding to a second public key $pk'$. Such a property, and stronger variants of it, are needed to ensure that *trial decryption* of anonymous ciphertexts does not produce unexpected results [30].

### 1.7.2 Key Encapsulation Mechanisms

A Key Encapsulation Mechanism (KEM) is a cryptographic scheme that simplifies the design and use of PKE. Whereas a PKE scheme can encrypt arbitrary messages, a KEM is limited to encrypting symmetric keys. One can then build a PKE scheme from a KEM and an AE scheme (called a Data Encapsulation Mechanism, DEM, in this context): first use the KEM to encrypt a symmetric key $K$, then use $K$ in the AE scheme to encrypt the desired message; ciphertexts now consist of two components: the encrypted key and the encrypted message.

We can define IND-CPA and IND-CCA security notions for KEMs. These are simpler to work with than the corresponding notions for PKE and this simplifies security analysis (i.e. generating and checking formal proofs). Moreover, there is a *composition theorem* for KEMs which says that if one takes an IND-CCA secure KEM and combines it with an AE-secure DEM (AE scheme) as above, then one gets an IND-CCA secure PKE scheme. As well as simplifying design and analysis, this KEM-DEM or hybrid viewpoint on PKE reflects how PKE is used in practice. Because PKE has generally slow algorithms and has large ciphertext overhead (compared to symmetric alternatives like AE), we do not use it directly to encrypt messages. Instead, we use a KEM to encrypt a short symmetric key and then use that key to encrypt our bulk messages.

### 1.7.3 Some common PKE schemes and KEMs

Perhaps the most famous PKE scheme is the RSA scheme. In its *textbook* form, the public key consists of a pair $(e, N)$ where $N$ is a product $p \cdot q$ of two large primes and the private key consists of a value $d$ such that $de = 1 \bmod (p-1)(q-1)$. Encryption of a message $M$, seen as a large integer modulo $N$, sets $C = M^e \bmod N$. On account of the mathematical relationship between $d$ and $e$, it can be shown that then $M = C^d \bmod N$. So encryption is done by "raising to the power of $e$ mod $N$" and decryption is done by "raising to the power of $d$ mod $N$". These operations can be carried out efficiently using the square-and-multiply algorithm and its variants. Decryption can be accelerated by working separately modulo $p$ and $q$ and then combining the results using the Chinese Remainder Theorem (CRT).

The security of RSA, informally, depends on the hardness of the Integer Factorisation Problem (IFP): if an adversary can recover $p$ and $q$ from $N$, then it can recompute $d$ from $e$, $p$ and $q$ using the extended Euclidean algorithm. But what we really want is a converse result: if an adversary can break RSA, then it should be possible to use that adversary in a black-box manner as a subroutine to create an algorithm that factors the modulus $N$ or solves some other presumed-to-be-hard problem.

This textbook version of RSA is completely insecure and must not be used in practice. Notice, for example, that it is not randomised, so it certainly cannot be IND-CPA secure. Instead RSA must be used as the basis for constructing more secure alternatives. This is usually done by performing a keyless encoding step, represented as a function $\mu(\cdot)$, on the message before applying the RSA transform. Thus we have $C = \mu(M)^e \bmod N$. Decryption then involves applying the reverse transform and decoding.

In order to achieve modern security notions, $\mu(\cdot)$ must be randomised. One popular encoding scheme, called PKCS#1 v1.5 and specified in [31], became very common in applications due to its relatively early standardisation and its ease of implementation. Unfortunately, RSA with PKCS#1 v1.5 encoding does not achieve IND-CCA security, as demonstrated by the famous Bleichenbacher attack [32]. Despite now being more than 20 years old, variants of the Bleichenbacher attack still regularly affect cryptographic deployments, see for example [33].

A better encoding scheme is provided in PKCS#1 v2.1 (also specified in [31]), but it has not fully displaced the earlier variant. RSA with PKCS#1 v2.1 encoding — also called RSA-OAEP — can be proven to yield an IND-CCA secure PKE scheme but the best security proof we have [34] is unsatisfactory in various technical respects: its proof is not tight and requires making a strong assumption about the hardness of a certain computational problem. Improved variants with better security analyses do exist in the literature but have not found their way into use.

One can easily build an IND-CCA secure KEM from the RSA primitive, as follows: select $M$ at random from $\{0, 1, \ldots, N-1\}$, set $C = M^e \bmod N$ (as in textbook RSA) and define $K = H(M)$ to be the encrypted symmetric key. Here $H$ is a cryptographic hash function (e.g. SHA-256). This scheme can be proven secure by modelling $H$ as a random oracle, under the assumption that the *RSA inversion problem* is hard. The RSA inversion problem is, informally, given $e$, $M$ and $M^e \bmod N$ for a random $M$, to recover $M$. The RSA inversion problem is not harder than the IFP, since any algorithm to solve IFP can be used to construct an algorithm that solves the RSA inversion problem. But the RSA inversion problem could be easier than the IFP and it is an open problem to fully decide this question.

Note that RSA encryption is gradually being displaced in applications by schemes using Elliptic Curve Cryptography (ECC) because of its superior performance and smaller key-sizes for a given target security level. See Section 1.13 for further discussion.

We will discuss another class of PKE schemes, based on the Discrete Logarithm Problem (DLP) after discussing Diffie-Hellman Key Exchange.

## 1.8 Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange (DHKE) is a fundamental tool in cryptography and was introduced by Diffie and Hellman in their seminal paper on Public Key Cryptography [35]. DHKE allows two parties to set up a shared key based only on a public exchange of messages. First the two parties (let us follow convention and call them Alice and Bob) agree on some common parameters: a group $G$ of prime order $q$ and a generator $g$ of that group. Typically, these are agreed during the exchange of messages or are pre-agreed; ideally, they are standardised so that well-vetted parameters of well understood cryptographic strength are used. Alice then chooses a value $x$ uniformly at random from $\{0, 1, \ldots, q-1\}$, computes $g^x$ using the group operation in $G$ and sends $g^x$ to Bob. Similarly Bob chooses a value $y$ uniformly at random from $\{0, 1, \ldots, q-1\}$, computes $g^y$ and sends it to Alice. Now Bob can compute $(g^x)^y = g^{xy}$ while Alice can compute $(g^y)^x = g^{yx} = g^{xy}$. Thus the value $g^{xy}$ is a common value that both Alice and Bob can compute.

An adversary Eve who eavesdrops on the communications between Alice and Bob can see $g^x$ and $g^y$ and can be assumed to know the public parameters $g, q$ and a description of the group $G$. This adversary is then faced with the problem of computing $g^{xy}$ from the triple $(g, g^x, g^y)$. Informally, this is known as the Computational Diffie-Hellman Problem (CDHP). One way for Eve to proceed is to try to compute $x$ from $g^x$ and then follow Alice's computation. Computing $x$ from $g$ and $g^x$ is known as the Discrete Logarithm Problem (DLP). The CHDP is not harder than the DLP (since an algorithm to solve the latter can be used to solve the former) but the exact relationship is not known.

The traditional setting for DHKE is to select large primes $p$ and $q$ such that $q$ divides $p-1$ and then take $g$ to be an element of multiplicative order $q$ modulo $p$; such a $g$ generates a group of order $q$. By choosing $q$ and $p$ of an appropriate size, we can make the DLP, and presumably the

CDHP, hard enough to attain a desired security level (see further discussion in Section 1.13). An alternative that has largely now displaced this traditional “finite field Diffie-Hellman” setting in applications is to use as $G$ the group of points on an elliptic curve over a finite field. This allows more efficient implementation and smaller key sizes at high security levels, but comes with additional implementation pitfalls.

The raw DHKE protocol as described directly above is rarely used in practice, because it is vulnerable to active Man-in-the-Middle (MitM) attacks, in which the adversary replaces the values $g^x$ and $g^y$ exchanged in the protocol with values for which it knows the discrete logarithms. However, the core idea of doing “multiplication in the exponent” is used repeatedly in cryptographic protocols. MitM attacks are generally prevented by adding some form of authentication (via MACs or digital signatures) to the protocol. This leads to the notion of Authenticated Key Exchange (AKE) protocols — see [36] for a comprehensive treatment.

It is important also for Alice and Bob to use trusted parameters, or to verify the cryptographic strength of the parameters that they receive, in DHKE. This can be a complex undertaking even in the traditional setting, since a robust primality test is needed [37, 38]. In the elliptic curve setting, it is not reasonable to expect the verification to be done by protocol participants and we use one of a small number of standardised curves. It is also important for the respective parties to check that the received values $g^y$ and $g^x$ do lie in the expected group, otherwise the protocol may be subject to attacks such as the small sub-group attack. These checks may be computationally costly.

### 1.8.1 From Diffie-Hellman to ElGamal

It is easy to build a KEM from the DHKE primitive. We simply set the `KeyGen` algorithm to output a key pair $(sk, pk) = (y, g^y)$ (where $y$ is generated as in DHKE), while `Encrypt` selects $x$ as in DHKE and then simply outputs the group element $g^x$ as the ciphertext. Finally, `Decrypt` takes as input a group element $h$ and outputs KDF$(h^y)$ where KDF$(\cdot)$ denotes a suitable key derivation function (as covered in Section 3.2). So the symmetric key encapsulated by group element $g^x$ is KDF$(g^{xy})$.

From this KEM, using the standard KEM-DEM construction, we obtain a variant of the ElGamal encryption scheme [39] called the Diffie-Hellman Integrated Encryption Scheme (DHIES) and analysed in [40]. In the elliptic curve setting, the scheme is known as ECIES. It is a particularly neat PKE scheme with compact ciphertexts and strong security properties. It avoids many of implementation issues associated with standardised variants of RSA.

## 1.9 Digital Signatures

Digital signatures schemes are used to provide authentication, integrity and non-repudiation services. They are the asymmetric analogue of MACs. A digital signature scheme consists of three algorithms: `KeyGen`, `Sign` and `Verify`. The first, `KeyGen`, is responsible for key generation. It is randomised and outputs key pairs $(sk, vk)$ where $sk$ denotes a private key (often called the secret key or signing key) and $vk$ denotes a verification key. The second algorithm `Sign` takes as input $sk$ and a message $m$ encoded as a bit-string and produces a signature $\sigma$, usually a short string of some fixed length $t$. The third algorithm is called `Verify`. This algorithm is used to verify the validity of a message/signature combination $(m, \sigma)$ under the key $vk$. So `Verify` takes as input triples $(vk, m, \sigma)$ and produces a binary output, with “1” indicating validity of the input triple.

The main security property required of a digital signature scheme is that it should be hard for an adversary to come up with a new pair $(m, \sigma)$ which `Verify` accepts with key $vk$, even when the adversary has already seen many pairs $(m_1, \sigma_1), (m_2, \sigma_2), \ldots$ produced by `Sign` with the matching signing key $sk$ for messages of its choice $m_1, m_2, \ldots$. As for MACs, the formal security notion is called Strong Unforgeability under Chosen Message Attack (SUF-CMA for short).

SUF-CMA digital signature schemes can be used to provide data origin authentication and data integrity services as per MACs. In addition, they can offer *non-repudiation*: if Alice is known to be associated with a key pair $(sk, vk)$ and provided $sk$ has not been compromised, then Alice cannot deny having created a valid message/signature pair $(m, \sigma)$ which `Verify` accepts. In practice, making this non-repudiation property meaningfully binding (e.g. for it to have legal force) is difficult. See further discussion in Law & Regulation CyBOK Knowledge Area [9].

A common pitfall is to assume that a signature $\sigma$ must bind a message $m$ and a verification key $vk$; that is, if `Verify` accepts on input $(vk, m, \sigma)$, then this implies that $\sigma$ must have been produced using the corresponding signing key $sk$ on message $m$. In fact, the SUF-CMA security definition does not imply this, as it only refers to security under a single key pair $(sk, vk)$ but does not rule out the possibility that, given as input a valid triple $(vk, m, \sigma)$, an adversary can concoct an alternative key pair $(sk', vk')$ such that `Verify` also accepts on input $(vk', m, \sigma)$. This gap leads to Duplicate Signature Key Selection (DSKS) attacks, see [41] for a detailed treatment. Such attacks can lead to serious vulnerabilities when using digital signatures in more complex protocols.

Signature schemes can be built from the same kinds of mathematics as PKE schemes. For example, the textbook RSA signature scheme signs a message $M$ by computing $\sigma = H(m)^d \bmod N$ where $H$ is a cryptographic hash function. Here $d$ is the signing key and the verification key is a pair $(e, N = pq)$ such that $de = 1 \bmod (p-1)(q-1)$. Then verification of a purported signature $\sigma$ on a message $m$ involves checking the equality $\sigma^e = H(m) \bmod N$. The similarity to RSA encryption, wherein signing uses the same operation of "raising to the power $d$ mod $N$" as does decryption in textbook RSA, is not coincidental. It is also a source of much confusion, since in the case of general signature schemes, signing is not related to any PKE decryption operation. A variation of textbook RSA in which $H(m)$ is replaced by a randomised encoding of the hashed message according to the PKCS#1 version 1.5 encoding scheme for signatures [31] is widely used in practice but has significant security shortcomings and lacks a formal security proof. The RSA-PSS encoding scheme, another randomised variant, is also specified in [31]; it does permit a formal security proof of security [42].

The DSA scheme [43] works in the finite field Diffie-Hellman setting, while ECDSA translates DSA to the elliptic curve setting. Notably, despite their standardised form and widespread use, neither DSA nor ECDSA enjoys a fully satisfactory UF-CMA security proof. The EdDSA signature scheme [44] is a variant of ECDSA that arguably enjoys better implementation security than ECDSA. In particular, ECDSA has a randomised signing algorithm and its security fails spectacularly (allowing recovery of the signing key) if the same random value is used to sign two different messages; ECDSA is also vulnerable to attacks exploiting partial leakage of the random values used during signing. By contrast, EdDSA is a deterministic scheme and avoids the worst of these failure modes. EdDSA, being more closely based on Schnorr signatures than ECDSA, also enjoys security proofs based on assumptions that are milder than those needed for proving the security of ECDSA.

Interestingly, signature schemes can be built from symmetric primitives, specifically hash functions. The original idea goes back to Lamport [45]: to be able to sign a single bit message, commit in the verification key to two values $h_0 = H(M_0)$ and $h_1 = H(M_1)$, where $M_0$ encodes a zero-bit and $M_1$ encodes a one-bit. So the verification key is $(h_0, h_1)$ and the signing key is $(M_0, M_1)$. Now to sign a single bit $b$, the signer simply outputs $M_b$; the verification algorithm checks the relation $h_b = H(M_b)$, outputting "1" if this holds. The original Lamport scheme is one-time use only and only signs a single-bit message. But many enhancements have been made to it over the years bringing it to a practically usable state. A specific hash-based signature scheme SPHINCS+ is an "alternate candidate" in the NIST PQC process for selecting post-quantum secure schemes, see Section 1.16 for further discussion.

Many forms of signature scheme with advanced security properties have been researched and sometimes find their way into use, especially in privacy-oriented applications. For example, *blind* signatures [46] allow a party to obtain signatures on messages without the signer knowing which message is being signed. Blind signature schemes can be used as a building block in electronic cash and electronic voting schemes. As a second example, *group* signatures allow one of many parties to sign messages in such a way that the verifier cannot tell exactly which party produced the signature; meanwhile a group manager can "break" this anonymity property. Group signatures have been used in the Trusted Computing Group's Direct Anonymous Attestation protocol [47] to enable remote authentication of Trusted Platform Modules whilst preserving privacy. A third example is provided by *ring* signatures [48], which have functionality similar to group signatures but without the opening capability possessed by a group manager. The cryptocurrency Monero has used ring signatures to provide anonymity.

## 1.10 Cryptographic Diversity

In the preceding subsections we have focused on the main algorithms and schemes that will be encountered in today's deployed cryptographic systems. However, a brief glance at the literature will reveal that this is just the tip of the iceberg in terms of what cryptographic ideas have been researched. But the vast majority of these ideas are not standardised, or do not have readily available implementations of production quality, or are not fully fleshed out in a way that a software engineer could easily implement them. There is a long road from cryptography as presented in most research papers and cryptography as it needs to be packaged for easy deployment.

## 1.11 The Adversary

So far, we have referred to the adversary as an abstract entity. In practice there are adversaries with different motivations, levels of skill and available computational resources. In applied cryptography we generally want to design cryptographic components that are usable across a wide variety of applications, so we should be conservative in how we model the adversary, assuming it is capable of marshalling significant resources.

At the same time, for the most part, we cannot achieve *unconditional security* in practical systems, that is security against adversaries with unbounded computational power. This is because such systems consume large amounts of key material that cannot be established using public key methods. So we have to place some limits on the adversary's computational capabilities. A typical objective is to make the adversary expend work comparable to an exhaustive key search for a block cipher like AES with 128-bit keys, in which case we speak of a “128-bit security level”.[8] Such a computational feat seems still well beyond the horizon of even state security agencies. It’s naturally hard to estimate their capabilities but, for comparison, the number of hash computations carried out in global Bitcoin mining currently stands at around $2^{67}$ per second and has the electricity consumption of a small sovereign state. At that rate, and assuming the cost of computing a hash is the same as that of testing an AES key, an exhaustive key search would still require $2^{36}$, or about $10^{11}$, years.[9] If we are even more conservative, or want a very large security margin over a long period of time (during which large-scale quantum computers may become available), or are concerned about concrete security in the context of multi-target attacks, then aiming for 192-bit or 256-bit security may be attractive.

We should also be conservative in rejecting algorithms and schemes that have known weaknesses, even if seemingly minor. It is a truism that attacks in cryptography only get stronger with time, either due to computational advances or the introduction of new cryptanalytic ideas. This conservatism is in tension with the fact that replacing one cryptographic scheme with another can be costly and time-consuming, unless *cryptographic agility* is built into our system (see Section 1.14 for further discussion). We are often encumbered with legacy cryptographic systems that cannot be easily updated, or where the system owners do not see the value in doing so until a practical break is exhibited.

## 1.12 The Role of Formal Security Definitions and Proofs

We have described in the preceding subsections, in an informal manner, syntax and security definitions for the main cryptographic schemes. These informal definitions are backed by fully formal ones, see for example [1, 2]. The value of such definitions are manifold. Syntax and correctness definitions enable one to be precise about what behaviour to expect from a cryptographic scheme and to build schemes out of simpler components in a precise way. Security definitions allow different schemes to be compared and to check whether application requirements will be met. Strong security definitions can rule out many classes of practical attack. A security proof — showing that a given scheme satisfies a given formal security definition under certain assumptions — then gives assurance as to the soundness of a scheme’s design and makes it clear what assumptions security rests on.

Indeed, formal security analysis in Applied Cryptography has reached a maturity level where there should be no need to ever deploy a cryptographic scheme or protocol that does *not* come with a clear syntax and have a rigorous security proof under clearly stated assumptions. Unfortunately, this rule is still often ignored in practice. This does leave opportunities for *post hoc* analysis by researchers, either to find flaws or to provide positive security results. In an ideal world, one would first gather all the requirements for any new scheme or protocol, then design the system and simultaneously develop security proofs for it. In reality, one is often trapped in a design-release-break-fix cycle. An intermediate approach of design-break-fix-release was used for TLS 1.3. Further discussion comparing these different models of cryptographic development can be found in [49].

[8] It is difficult to be precise about concrete security levels. Issues arise because of different cost models for computation, e.g. counting the unit of cost as being an AES operation versus a single CPU) instruction; on some platforms, an AES round operation can be performed via a single CPU instruction! One also needs to account for the use of CPUs, GPUs and special purpose hardware for cryptanalysis, such as might be within the reach of a well-funded security agency.

[9] It is worth noting that the sun is expected to consume the earth in a super nova in about $10^{10}$ years, assuming it is not swallowed by a black hole first.

Notice also that such proofs are not unconditional, unlike most proofs in mathematics. A typical proof shows that a given scheme or protocol satisfies a particular security definition under some assumptions. Such proofs are often stated in a reductive fashion (i.e. as with reductions from complexity theory): given any adversary in the form of an arbitrary algorithm that can break the scheme according to the security definition, the proof shows that the adversary $A$ can be used as a subroutine in building an algorithm $B$ that can break one of the components of the scheme (e.g. find a collision in a hash function) or in building a different algorithm $C$ that can break some underlying hardness assumption (e.g. solve the IFP for moduli $n$ with distribution given by the KeyGen algorithm of a PKE scheme). For Applied Cryptography, *concrete* reductions are to be preferred. In our example, these are ones in which in which we eschew statements describing $B$ or $C$ as simply being "polynomial time" but in which the resources (computation, storage, etc) consumed by the adversary $A$ (and its advantage in breaking the scheme) are carefully related to those of algorithms $B$ and $C$. Furthermore, it is preferable that proofs should be *tight*. That is, we would like to have proofs showing a close relationship between the resources consumed by and advantage of adversary $A$ on the one hand, and the resources consumed by and advantages of algorithms $B$ and $C$ constructed from $A$ on the other. The result of having a tight proof is that the scheme's security can be meaningfully related to that of its underlying components. This is not always achieved, resulting in proofs that may be technically vacuous.

For complex cryptographic schemes and protocols, the security statements can end up being difficult to interpret, as they may involve many terms and each term may relate to a different component in a non-trivial way. Such security statements typically arise from proofs with many hand-written steps that can hide errors or be difficult for humans to verify. Typically though, such proofs are modularised in a sequence of steps that can be individually checked and updated if found to be in error. A popular approach called "game hopping" or "sequences of games", as formalised in [50, 51] in two slightly different ways, lends itself to the generation of such proofs. An alternative approach to taming the complexity of proofs comes from the use of formal and automated analysis methods, see Formal Methods for Security CyBOK Knowledge Area [52] for an extensive treatment.

The proofs are usually for mathematically tractable pseudo-code descriptions of the schemes, not for the schemes as implemented in some high-level programming language and certainly not for schemes as implemented in a machine language. So there is a significant gap in terms of what artefacts the proofs actually make statements about. Researchers have had some success in developing tools that can prove the security of running code and some code of this type has been deployed in practice; for a good overview, see [53]. Furthermore, a security proof only gives guarantees concerning the success of attacks that lie within the scope of the model and says nothing about what happens beyond that. For example, an adversary operating in the real world may have greater capabilities than are provided to it in the security model used for proving security. We shall return to these issues in the next section on cryptographic implementation.

A sustained critique of the provable security approach has been mounted by Koblitz and Menezes in their "Another look at . . ." series of papers, see [54] for a retrospective. This critique has not always been welcomed by the theoretical cryptography research community, but any serious field should be able to sustain, reflect on and adapt to such critique. In our view, the work of Koblitz and Menezes has helped to bridge the gap between theory and practice in cryptography, since it has helped the community to understand and begin to address the limitations of its formal foundations.

## 1.13 Key Sizes

We have discussed why aiming for the 128-bit security level makes sense — it provides a sufficient margin of security in almost every conceivable circumstance, at least within the realm of conventional computing. Another reason is that except in specific application domains such as constrained environments, it is efficiently achievable. In other words, there is no good reason *not* to aim this high.

Algorithms and their keys do have finite lifetimes. Advances in cryptanalysis may render once secure algorithms or key sizes insecure. Moreover, the longer an individual key is in use, the more likely it is to become compromised. These issues for individual keys are discussed in more detail in Section 3. Changing algorithms and key sizes can be inconvenient and costly. This provides arguments in favour of making conservative choices in the first place.

The target security level of 128 bits brings efficiency considerations into play, especially for asymmetric algorithms. For example, it is estimated that forcing a direct attack on the IFP to break RSA cost $2^{128}$ effort would require the use of a 3072-bit modulus [55, Table 2].[10] This is because of the sub-exponential complexity of the best known algorithm for solving the IFP (the Number Field Sieve). Such a modulus size is large enough to significantly slow down the basic RSA operations (in the general case, the complexity of modular exponentiation grows with the cube of the modulus bit length). The same is true for finite-field DLP-based cryptosystems (e.g. Diffie-Hellman and ElGamal). By contrast, because only square-root speed-ups exist for the ECDLP, we can escape with much smaller parameters when targeting 128-bit security for ECC: a curve over a 256-bit prime field suffices. So, for the standard 128-bit security level, ECC-based schemes become more efficient than IFP-based or finite field DLP-based ones. The contrast is even more stark if one targets a 256-bit security level: there 15360-bit RSA public keys are recommended by NIST [55, Table 2], while the required size for ECC only moves up to 512 bits.

These considerations explain the recent rise in popularity of ECC and may lead to the slow death of RSA-based cryptosystems. The US National Security Agency (NSA) has recommended organisations who have not already done so to not make a significant expenditure to transition from RSA to ECC, but to wait for post-quantum algorithms (i.e. algorithms that aim to be secure against large-scale quantum computers) that should result from the on-going NIST standardisation effort.[11]

## 1.14 Cryptographic Agility

Occasionally it is necessary in some system or protocol to exchange one algorithm for another in the same class. One reason might be that the original algorithm is broken. The history of hash functions provides notable examples, with once secure hash functions like MD5 now being considered trivially insecure. Another reason might be that a more efficient alternative becomes available — consider the switch from RSA-based algorithms to ECC-based ones discussed in Section 1.13. A third reason is the introduction of a technology shift — for example, a precautionary shift to post-quantum algorithms as a hedge against the development of large-scale quantum computers.

This exchange is made easier if the system or protocol is cryptographically agile — that is,

[10] Other estimates are available, see summary at **https://www.keylength.com/en/**, but all estimates are in the same ballpark.

[11] See **https://apps.nsa.gov/iad/programs/iad-initiatives/cnsa-suite.cfm**.

if it has an in-built capability to switch one algorithm for another and/or from one version to another. This facility is enabled in secure communications protocols like IPsec, SSH and SSL/TLS through cipher suite and version negotiation: the algorithms that will be used and the protocol version are negotiated between the participating parties during the protocol execution itself. Adding this facility to an already complex protocol may introduce security vulnerabilities, since the negotiation mechanisms themselves may become a point of weakness. An example of this is downgrade attacks in the context of SSL/TLS, which have exploited the co-existence of different protocol versions [56] as well as support for deliberately weakened "EXPORT" cipher suites [57, 58]. Cryptographic agility may also induce software bloat as there is an incipient temptation to add everyone's favourite algorithm.

At the opposite end of the spectrum from cryptographic agility lies systems (and their designers) that are *cryptographically opinionated*, that is, where a single set of algorithms is selected and hard-coded. WireGuard [59] is an example of such a protocol: it has no facility to change algorithms and not even a protocol version field.

There is a middle-way: support cryptographic agility where possible, but with tight control over which algorithms and legacy versions are supported.

For more information on cryptographic agility, especially in the post-quantum setting, we recommend [60].

## 1.15 Development of Standardised Cryptography

Standardisation plays an important role in cryptography. Standards provide a suite of carefully vetted primitives, higher-level protocols and cryptographic best practices that can be used by non-expert developers. They can also help to ensure interoperability, not only by providing complete specifications of algorithms but also by helping to identify a smaller set of algorithms on which implementers can focus.

There are multiple standardisation bodies producing cryptographic standards. Most prominent are ISO/IEC, the US National Institute for Standards and Technology (NIST) and the Internet Engineering Task Force (IETF). ISO/IEC and NIST cryptographic standards tend to focus (though not exclusively) on lower-level primitives, while IETF works more at the protocol level.

The three bodies work in quite different ways.

ISO/IEC uses a closed model, where country representatives come together to produce standards through a multi-stage drafting and voting process. ISO/IEC working groups can and do invite external experts to attend their meetings and provide input.

NIST employees directly write some of their standards, with open calls for comment from the wider community. NIST also runs cryptographic competitions in which external teams of researchers compete to meet a design specification. The AES and SHA-3 algorithms were produced in this way. Although NIST is a US federal government body, its standards tend to become *de facto* international standards. Its competitions are also entered by teams from all around the world and the winners are frequently not from the US.

The IETF model is completely open. Anyone can join an IETF mailing list and join a technical discussion or, given financial resources, attend IETF meetings where its standards are developed. For a document to become an IETF standard (technically, a "Request for Comments" or RFC), the key requirement is "rough consensus and running code". Multiple levels of review and consensus-building are involved before a draft document becomes an RFC. The Internet

Research Task Force (IRTF) is a sister-organisation to the IETF and its Crypto Forum Research Group (CFRG)[12] acts as a repository of expertise on which the IETF can draw. CFRG also produces its own RFCs.

Standards bodies are not perfect. Too many bodies — and standards produced by them — can lead to cryptographic proliferation, which makes inter-operation harder to achieve.They can also lead to subtle incompatibilities between different versions of the same algorithms. Even completely open standards bodies may fail to gather input from the right set of stakeholders. Standards bodies can act prematurely and standardise a version of a scheme that is later found to be deficient in some way, or where improved options only emerge later. Once the standard is set, in the absence of serious attacks, there may be little incentive to change it. The history of PKE schemes based on RSA illustrates this well (see Section 1.7.3): RSA with PKCS#1 v1.5 encoding has led to many security issues and the introduction of attack-specific work-arounds; RSA with PKCS#1 v2.1 encoding (RSA-OAEP) has been standardised for many years but has not become widely used; meanwhile even better ways of turning RSA into a KEM or a PKE have been discovered but have not become mainstream.

Standards bodies are also subject to “regulatory capture”, whereby groups representing specific national or commercial interests have the potential to influence the work of a standards body. For example, NSA had a role in the design of the DES algorithm [61, pp. 232-233], and, on another occasion, supplied the overall design of the Dual_EC_DBRG pseudorandom generator that was specified in a NIST standard [62], along with certain critical parameters [63, p. 17]. In such contexts, transparency as to the role of any national or commercial stakeholders is key. For instance, NIST have reviewed their cryptographic standardisation process [63] to increase transparency and decrease reliance on external organisations.

Other standards bodies relevant for cryptography include ETSI (which is active in post-quantum cryptographic standardisation, as discussed immediately below) and IEEE (which developed an early series of standards for Public Key Cryptography, IEEE P1363).

## 1.16 Post-quantum Cryptography

Large-scale quantum computers, if they could be built, would severely threaten RSA-based and discrete-log-based cryptographic algorithms in both finite field and elliptic curve settings. This includes almost all of the Public Key Cryptography in use today. This is because of Shor’s algorithm [64], a quantum algorithm that leads to polynomial-time algorithms for both IFP and DLP in both finite field and the elliptic curve settings. This stands in strong contrast to the situation with the best known classical, non-quantum, algorithms for these problems which are super-polynomial, but sub-exponential for IFP and the DLP in finite fields (and fully exponential for the DLP in the elliptic curve setting). Quantum computers also have some consequences for symmetric algorithms due to Grover’s algorithm [65], which in theory provides a quadratic speed-up for exhaustive key search. However, the impact there is less substantial — as a rule of thumb, doubling the key size is sufficient to thwart quantum attacks.

Post-quantum cryptography (PQC) refers to the study of cryptographic algorithms and schemes that are plausibly secure against large-scale quantum algorithms. Such algorithms and schemes are still classical: they are designed to run on classical computers. We use the term “plausibly” here, since the field of quantum algorithms is still young and it is hard to anticipate future developments. Note that PQC is sometimes referred to as quantum-safe, quantum

[12] See **https://irtf.org/cfrg**.

resistant, or quantum-immune cryptography. PQC has been an active but niche research field for many years.

In late 2016, in response to projected progress in scaling quantum computing and recognising the long transition times needed for introducing new cryptographic schemes, NIST launched a process to define a suite of post-quantum schemes.[13] The focus of the NIST process is on KEMs and digital signature schemes, since the threat quantum computing poses for symmetric schemes is relatively weaker than it is for public key schemes. At the time of writing in mid 2021, the process (actually a competition) has entered its third round and a set of finalist schemes has been selected, alongside a set of alternate, or back-up schemes. The NIST process should result in new NIST standards in the mid 2020s.

It will be a significant challenge to integrate the new schemes into widely-used protocols and systems in a standardised way. This is because the NIST finalists have quite different (and usually worse) performance profiles, in terms of key sizes, ciphertext or signature size and computation requirements, from existing public key schemes. Work is underway to address this challenge in IETF and ETSI and some deployment experiments have been carried out for the TLS protocol by Google and CloudFlare.[14] It is likely that post-quantum schemes will initially be deployed in *hybrid* modes alongside classical public key algorithms, to mitigate against immaturity of implementation and security analysis. The recent deployment experiments used hybrid modes.

## 1.17 Quantum Key Distribution

PQC should be distinguished from quantum cryptography, which attempts to harness quantum effects to build secure cryptographic schemes. The most mature branch of quantum cryptography is Quantum Key Distribution (QKD). A QKD protocol typically uses polarised photons to transmit information from one party to another, in such a way that an eavesdropper trying to intercept the signals will disturb them in a detectable way. The two parties can engage in a resolution protocol over a classical, authenticated channel that leads to them agreeing on keying material about which even a computationally unbounded adversary has minimal information.

QKD is often marketed as offering unconditional security assuming only the correctness of the known laws of physics. In terms of commercial deployment, QKD faces severe challenges. The main one is that it does not solve a problem that we cannot solve satisfactorily using other means. Moreover, those means are already commoditised. Subsidiary issues are: the need for expensive special-purpose hardware, the need for an authentic channel (if we have such a channel, then we could use it to distribute public keys instead), limitations on range that stand in opposition to standard end-to-end security requirements, limitations on the rate at which secure keys can be established (leading to hybrid QKD/classical systems, thereby obviating any unconditional security guarantees), and the fact that theoretical guarantees of unconditional security are hard to translate into practice.

[13] See **https://csrc.nist.gov/projects/post-quantum-cryptography**.
[14] See **https://blog.cloudflare.com/the-tls-post-quantum-experiment**.

## 1.18 From Schemes to Protocols

In this section, we have focused on low-level cryptographic algorithms and schemes. In real systems, these are combined to build more complex systems. Often, the result is a collection of interactive algorithms, commonly called a cryptographic protocol. An overall cryptographic system may use a number of sub-systems and protocols. We will look briefly at a few specific systems in Section 5. Here, we restrict ourselves to general comments about such systems.

First, to reiterate from Section 1.12, even complex cryptographic protocols and systems, and their security properties, are amenable to rigorous definitions and analysis. The approach is to analyse the security of such systems in terms of simpler, easier to analyse security properties of their components. We have seen a simple example of this in our treatment of AE Section 1.6 in, where a generic composition approach allows the AE(AD) security of the EtM composition to be established based on the IND-CPA security of its “E” component and the SUF-CMA security of its “M” component.

This process could be carried to the next level. Consider, for example, building a unidirectional secure channel protocol, assuming a suitable symmetric key is already in place at the sender and receiver. First we need a definition of what functionality and security such a protocol should provide. For example, we could demand integrity and confidentiality of plaintexts sent and that an adversary that tries to reorder, drop, or replay ciphertexts can be detected. Suitable formal definitions capturing these requirements can be found in [66].

Then we can try to realise the unidirectional secure channel protocol from a nonce-based AEAD scheme and prove that its security follows from (or can be reduced to) the standard security definition for AEAD security. Here, a candidate construction is to make the sender stateful, by having it maintain a counter for the number of plaintexts encrypted. That counter is encoded as the nonce for the AEAD scheme. The receiver maintains an independent copy of the counter, using it as the nonce when performing decryption. Intuitively, confidentiality and integrity for individual ciphertexts in the secure channel follows immediately from the AEAD security definition. Meanwhile, an adversary tampering with the order of ciphertexts will lead to the receiver using the wrong counter value when decrypting, which leads to an error by the integrity properties of the AEAD scheme. These ideas can and should be formalised.

We can go even further and build a bidirectional secure channel protocol from a unidirectional one. Here, additional considerations arise from the possibility of reflection attacks and whether the channel should preserve the joint ordering of ciphertexts in both directions. We can also consider secure channel protocols in which the protocol recovers from accidental packet losses or reordering arising from the underlying network transport, or where such errors are fatal and lead to termination of the protocol. We can try to remove the assumption of having pre-established symmetric keys by bringing a key exchange component into play as a separate or integrated sub-protocol.

Again, all these aspects can be formally defined and analysed. However, the challenges in dealing with the complexity inherent in such systems should already be apparent, especially when the models and proofs are all hand-generated. For this reason, it is common to make some kind of “cryptographic core” of a system the focus of analysis and to abstract away many details. For example, a typical analysis will completely ignore all key management aspects, including PKI (which we discuss in Section 3.8). Instead, it is simply assumed that all keys are authentic and where they need to be, as we did in the example above. However, these details are relevant to the overall security of the system. So too much abstraction brings the risk of missing important facets or making assumptions that are not warranted in practice.

It is then important to be clear about what is — and is not — being formally specified and analysed.

An alternative approach to taming complexity is to use mechanised tools, letting a computer do the heavy lifting. However, the currently available tools are quite difficult to use and require human intervention and, more often than not, input from the tool designer. One of the more successful approaches here is to use a symbolic model of the cryptographic primitives rather than a computational one as we have been considering so far. This provides a level of abstraction that enables more complex protocols to be considered, but which misses some of the subtleties of the computational approach. Symbolic approaches to formal analysis are covered in more detail in Formal Methods for Security CyBOK Knowledge Area [52].

## 2 CRYPTOGRAPHIC IMPLEMENTATION

[25, 67, 68, 69]

So far, we have focused on describing cryptographic algorithms and schemes in terms of abstract algorithms or in mathematical terms. In research papers, new schemes are usually presented as pseudo-code. Of course, this all needs to be translated into actual code (or hardware) for practical use. In this section, we briefly discuss some of the considerations that arise in this process.

### 2.1 Cryptographic Libraries

From the perspective of the developer, cryptography is usually consumed via a cryptographic library, that is, a collection of algorithms and schemes accessed through an API. Many different cryptographic libraries are available, in many different programming languages, though ‘C’ and Java libraries are most common. Different libraries have different licence restrictions, though many are available under non-restrictive “open source” licences of various kinds.

Some libraries (e.g. OpenSSL[15] or BouncyCastle[16]) are richly featured, supporting many different cryptographic schemes and processes and can be used across a wide range of applications. Others are much more restrictive and designed to support only certain use cases. In part, this reflects the taste of the libraries’ authors, but also age and available development resources.

Some libraries are better maintained than others. For example, prior to the Heartbleed vulnerability (discussed in Section 3.5), OpenSSL had fallen into a state of some ossification and disrepair. Consequently, Google and OpenBSD separately decided to “fork” OpenSSL, that is to create entirely separate development branches of the library, resulting in the BoringSSL and LibreSSL libraries. Heartbleed also resulted in a broader realisation of how important OpenSSL was to the whole Internet ecosystem. A sequence of reforms of the OpenSSL project followed and today the project is in much better shape, with a larger team of core developers, better funding and more active development.

Some cryptographic libraries are developed by professional software engineers with considerable experience in avoiding some of the pitfalls we discuss in this section and elsewhere. Others are not. In many cases, the developers are working on a volunteer basis; most of

[15]https://www.openssl.org/
[16]https://www.bouncycastle.org/

OpenSSL's code development is done in this way. As part of its support model, a cryptographic library should have a clear process for notifying its maintainers of bugs and security vulnerabilities. The library's developers should commit to address these in a timely manner.

## 2.2 API Design for Cryptographic Libraries

The API that a cryptographic library presents to its consumers is critical. There is a delicate balance to be struck between flexibility (allowing developers to use the library in a wide variety of ways, thereby making it more useful) and security (restricting the API in an effort to prevent developers from using the library in insecure ways). Consider the problem of providing an API for symmetric encryption. Should the library allow direct access to a raw block cipher capability? Possibly, since some developers may need that functionality at some point. But also perhaps not — since it's probable that an inexperienced developer will use the block cipher in ECB mode to perform bulk encryption, with predictably insecure results.[17] This simple example is not an isolated one. It could be replaced, for example, with one involving nonces in an API for AEAD, or one involving the selection of parameters for a primality test.

Green and Smith [67] present ten principles for API design for cryptographic libraries. Their principles are derived empirically from their analysis of an *ad hoc* collection of examples and from interviews with developers. More systematic approaches, relying on standard methodologies from social science, have followed, see [70] for an illustrative example of this line of work.

Green and Smith observe that developers' mistakes affect many users, so it makes sense to focus on them and not the actual end users, who are typically the target of usable security research. They point out that cryptographic libraries appear to be uniquely prone to misuse by developers, with even subtle misuse leading to catastrophic security failures. They also argue that as the use of cryptography in applications becomes more common, so cryptographic libraries are increasingly used by developers without cryptographic expertise.

Green and Smith's ten principles can be summarised as follows:

1. Integrate cryptographic functionality into standard APIs; that is, hide cryptography from developers where possible.
2. Make APIs sufficiently powerful to satisfy both security and non-security requirements. The argument here is that developers ultimately don't have a security goal in mind and satisfying their actual requirements, ascertained through interviewing them, will encourage them to use an API rather than writing their own cryptographic code.
3. Design APIs that are easy to learn without cryptographic expertise.
4. Don't break the developer's paradigm (or mental model) of what the API should look like.
5. Design APIs that are easy to use even without reading the documentation (since developers will not read it!).
6. Create APIs that are hard to misuse — visible errors should result from incorrect usage.
7. APIs should have safe and unambiguous defaults.
8. APIs should have testing modes, because otherwise developers will hack the API to turn off security during development to ease testing, but then may fail to properly remove

[17]See **https://blog.filippo.io/the-ecb-penguin/** for a vivid illustration of the limitations of ECB mode.

their hacks. An issue here is that the resulting code could be released with the testing mode still enabled, but one would hope that regular software assurance would detect this before release.

9. Code that uses the API should be easy to read and maintain. For example, iteration counts for password hashing should not be set by a developer via the API, but instead internally in the library. One issue here is that the internal defaults may be overkill and hurt performance in some use cases. This relates to the tension between flexibility and security.
10. The API should assist with or handle end user interaction, rather than leave the entire burden of this to the developer using the API. Here, error messages are highlighted as a particular concern by Green and Smith: the API and the library documentation should help developers understand what failure modes the library has, what the security consequences of these are and how the resulting errors should be handled by the calling code.

For additional references and discussion, see Human Factors CyBOK Knowledge Area [71].

## 2.3 Implementation Challenges

Having discussed libraries and their APIs, we now turn to challenges arising in securely implementing the algorithms and schemes within these libraries.

The main problem is to translate a purely mathematical or pseudo-code description of a scheme (the typical unit of analysis in formal security proofs arising in research papers) into running code on a real computer in such a way that the abstraction level involved in the security analysis is still properly respected by the running code. Put another way, the challenge is to ensure there are no mechanisms through which sensitive information can leak that are not already anticipated and eliminated by the security analysis. There are multiple ways in which such leakage can arise. We consider a representative selection here.

### 2.3.1 Length Side Channels

As we noted in Section 1.6, the usual security goal of an AEAD scheme does not guarantee that the length of plaintexts will be hidden. Indeed, AEAD schemes like AES-GCM make it trivial to read off the plaintext length from the ciphertext length. However, it is clear that length leakage can be fatal to security. Consider a simplistic secure trading system where a user issues only two commands, “BUY” or “SELL”, with these commands being encoded in simple ASCII and sent over a network under the protection of AES-GCM encryption. An adversary sitting on the network who can intercept the encrypted communications can trivially infer what commands a user is issuing, just by looking at ciphertext lengths (the ciphertexts for “SELL” will be one byte longer than those for “BUY”). More generally, attacks based on *traffic analysis* and on the analysis of metadata associated with encrypted data can result in significant information leaking to an adversary.

### 2.3.2 Timing Side Channels

The amount of time that it takes to execute the cryptographic code may leak information about the internal processing steps of the algorithm. This may in turn leak sensitive information, e.g. information about keys. The first public demonstration of this problem was made by Kocher [68] with the attacker having direct access to timing information. Later it was shown that such attacks were even feasible remotely, i.e. could be carried out by an attacker located at a different network location from the target, with timing information being polluted by network noise [72].

Consider for example a naive elliptic curve scalar multiplication routine which is optimised to ignore leading zeros in the most significant bits of the scalar. Here we imagine the scalar multiplication performing doubling and adding operations on points, with the operations being determined by the bits of the scalar from most significant to least significant. If the adversary can somehow time the execution of the scalar multiplication routine, it can detect cases where the code finishes early and infer which scalars have some number of most significant bits equal to zero. Depending on how the routine is used, this may provide enough side channel information to enable a key to be recovered. This is the case, for example, for the ECDSA scheme, where even partial leakage of random values can be exploited. Recent systematic studies in this specific setting [73, 74] show that timing attacks are still pertinent today.

### 2.3.3 Error Side Channels

Errors arising during cryptographic processing can also leak information about internal processing steps. Padding oracle attacks on CBC mode encryption, originally introduced in [24], provide a classic and persistent example of this phenomenon. CBC mode uses a block cipher to encrypt plaintext data that is a multiple of the block cipher's block length. But in applications, we typically want to encrypt data of arbitrary length. This implies that data needs to be padded to a block boundary of the block cipher before it can be encrypted by CBC mode. Vaudenay observed that, during decryption, this padding needs to be removed, but the padding may be invalidly formatted and the decryption code may produce an error message in this case. If the adversary can somehow observe the error message, then it can infer something about the padding's validity. By carefully constructing ciphertexts and observing errors arising during their decryption, an adversary can mount a plaintext recovery attack via this error side channel.

In practice, the error messages may themselves be encrypted, but then revealed via a secondary side channel, e.g. a timing side channel (since an implementation might abort further processing once a padding error is encountered). For examples of this in the context of SSL/TLS and which illustrate the difficulty of removing this class of side channel, see [75, 25].

### 2.3.4 Attacks Arising from Shared Resources

The cryptographic code may not be running in perfect isolation from potential adversaries. In particular, in modern CPUs), there is a memory cache hierarchy in which the same fast memory is shared between different processes, with each process potentially overwriting portions of the cache used by other processes. For example, in a cloud computing scenario, many different users' processes may be running in parallel on the same underlying hardware, even if they are separated by security techniques like virtualisation. So an attacker, running in a separate process in the CPU, could selectively flush portions of the cache and then, after the victim process has run some critical code, observe by timing its own cache accesses, whether that part of the cache has been accessed by the victim process or not. If the victim process has a pattern of memory access that is key-dependent, then this may indirectly leak information about the victim's key. This particular attack is known as a Flush+Reload attack and was introduced in [76]; several other forms of cache-based attack are known. The possibility of such attacks was first introduced in [77]; later such attacks were shown to be problematic for AES in particular [78].[18] In the last few years, researchers have had a field day developing cache-based and related micro-architectural attacks against cryptographic implementations. These attacks arise in general from designers of modern CPUs making architectural compromises in search of speed.

### 2.3.5 Implementation Weaknesses

More prosaically, cryptographic keys may be improperly deleted after use, or accidentally written to backup media. Plaintext may be improperly released to a calling application before its integrity has been verified. This can occur in certain constructions where MAC verification is done after decryption and also in streaming applications where only a limited-size buffer is available for holding decrypted data.

### 2.3.6 Attacks Arising from Composition

A system making use of multiple cryptographic components may inadvertently leak sensitive information through incorrect composition of those components. So we have leakage at a system level rather than directly from the individual cryptographic components. Consider the case of Zcash,[19] an anonymous cryptocurrency. Zcash uses a combination of zero-knowledge proofs, a PKE scheme and a commitment scheme in its transaction format. The PKE scheme is used as an outer layer and is anonymous, so the identity of the intended recipient is shielded. How then should a Zcash client decide if a transaction is intended for it? It has to perform a trial decryption using its private key; if this fails, no further processing is carried out. Otherwise, if decryption succeeds, then further cryptographic processing is done (e.g. the commitment is checked). This creates a potentially observable difference in behaviour that breaks the intended anonymity properties of Zcash [79]. The PKE scheme used may be IND-CCA secure and anonymous, but these atomic security properties do not suffice if the overall system's behaviour leaks the critical information.

### 2.3.7 Hardware Side Channels

Cryptography is often implemented directly in hardware. For example, hardware acceleration of cryptographic functions was once common, both in low-cost environments such as payment cards and in higher-end applications, such as server-side SSL/TLS operations. Today, Internet-of-Things (IoT) deployments may use hardware components to implement expensive cryptographic functions. Hardware-based cryptography can also be found in Trusted Platform Modules (TPMs) as specified by the Trusted Computing Group and in systems like Intel Software Guard eXtensions (SGX) and ARM Trustzone. As noted in Section 1.3, modern CPUs) have instructions to enable high performance implementation of important cryptographic algorithms like AES.

[18] See also **https://cr.yp.to/antiforgery/cachetiming-20050414.pdf** for contemporaneous but unpublished work.
[19] See **https://z.cash/**.

There are additional sources of leakage in hardware implementations of cryptography. For example, an attacker against a smartcard might be able to observe how much power the smartcard draws while carrying out its cryptographic operations at a fine-grained time resolution and this might reveal the type of operation being carried out at each moment in time. To give a more specific example, in an implementation of RSA decryption using a basic "square and multiply" approach, the two possible operations for each private key bit — either square or square & multiply — could consume different amounts of power and thus the private key can be read off bit-by-bit from a power trace. The electromagnetic emissions from a hardware implementation might also leak sensitive information. Even sonic side channels are possible. For example the first working QKD prototype [80] reportedly had such a side channel, since an observer could listen to the optical components physically moving and thereby learn which polarisation was being used for each signal being sent. This highlights just one of the many challenges in achieving unconditional security according to the laws of physics.

For a fuller discussion of hardware side channels, we refer the reader to Hardware Security CyBOK Knowledge Area [12].

### 2.3.8 Fault Attacks

Hardware implementations may also be vulnerable to fault or glitch attacks, where an error is introduced into cryptographic computations at a precise moment resulting in leakage of sensitive data (typically keys) via the output of the computation. The first such attack focused on implementations of RSA using the CRT [81]. A more recent incarnation of this form of attack called Rowhammer targets the induction of faults in memory locations where keys are stored by repeatedly writing to adjacent locations [82].

## 2.4 Defences

General techniques for defending against cryptographic implementation vulnerabilities (as opposed to weaknesses in the algorithms and schemes themselves) come from the fields of software and hardware security and are well-summarised in [83, 12]. Indeed, it can be argued that conventional software security may be more important for cryptographic code than for other forms of code. For hardware, blinding, masking, threshold techniques and physical shielding are commonly used protections. For software, common techniques include formal specification and verification of software and hardware designs, static and dynamic analysis of code, fuzzing, information flow analysis, the use of domain-specific languages for generating cryptographic code and the use of strong typing to model and enforce security properties. Most of the software techniques are currently supported only by experimental tools and are not at present widely deployed in production environments. Additionally, the objects they analyse — and therefore the protections they offer — only extend so far, down to code at Instruction Set Architecture level at best.

Length side channels can be closed by padding plaintexts to one of a set of predetermined sizes before encryption and by adding cover or dummy traffic. Secure communications protocols like SSL/TLS and IPsec have features supporting such operations, but these features are not widely used in practice.

A set of coding practices aim to achieve what is loosely called *Constant-Time Cryptography*. The core idea is to remove, through careful programming, any correlation between the values of sensitive data such as keys or plaintexts, and variables that can be observed by an adversary such as execution time. This entails avoiding, amongst other things, key-dependent memory accesses, key-dependent branching and certain low-level instructions whose running time is operand-dependent. It may also require writing high-level code in particular ways so as to prevent the compiler from optimising away constant-time protections.[20] Writing constant-time code for existing algorithms is non-trivial. In some cases, cryptographic designers have taken it into account from the beginning when designing their algorithms. For example, Bernstein's ChaCha20 algorithm[21] does so, while using certain coordinate systems makes it easier to achieve constant-time implementation of elliptic curve algorithms [84].

## 2.5 Random Bit Generation

Cryptography relies on randomness in a crucial way. Most obviously, random bits are needed for symmetric keys, and for more complex key generation algorithms in the public key setting. But to achieve standard security notions such as IND-CPA security, PKE schemes need to have a randomised encryption algorithm. Fortunately, in the nonce-based AE setting, we can avoid the need for randomness during encryption. Some signature schemes have a randomised signing algorithm. This is the case for RSA PSS, DSA and ECDSA, for example.[22]

A failure to supply suitable randomness to such algorithms can have disastrous consequences. We already remarked on this in the context of DSA and ECDSA in Section 1.9. We will discuss examples for asymmetric key pair generation in Section 3.4.

So our cryptographic algorithms need to have access to "strong" random bit sources. To be generally applicable, such a source should offer a plentiful supply of bits that are independent and uniformly distributed, such that the adversary has no information about them. Specific algorithms may reveal their random bits, but general usage requires that they remain hidden.

In an ideal world, every computing device would be equipped with a True Random Bit Generator (TRBG)[23] whose output is hidden from potential adversaries. In practice, this has proven to be very difficult to achieve. Intel and AMD CPUs) do offer access to the post-processed output of a TRBG via the RDRAND instruction. However, the designs of these TRBGs are not fully open.

In the absence of a TRBG, common practice is for the operating system to gather data from weak, local entropy sources such as keyboard timings, disk access times, process IDs and packet arrival times, to mix this data together in a so-called *entropy pool* and then to extract pseudo-random bits from the pool as needed using a suitable cryptographic function (a Pseudo-Random Number Generator, PRNG, using a seed derived from the entropy pool). Designs of this type are standardised by NIST in [62]. They are also used in most operating systems but with a variety of *ad hoc* and hard-to-analyse constructions. Mature formal security models and constructions for random bit generators do exist, see [69] for a survey. But this is yet another instance where practice initially got ahead of theory, then useful theory was developed, and now practice is yet to fully catch up again.

It is challenging to estimate how much true randomness can be gathered from the aforementioned weak entropy sources. In some computing environments, such as embedded systems, some or all of the sources may be absent, leading to slow filling of the entropy pool after a reboot — leaving a "boot time entropy hole" [86, 87]. A related issue arises in Virtual Machine

[20] An introduction to the paradigm can be found at **https://www.bearssl.org/constanttime.html**.
[21] See **https://cr.yp.to/chacha.html**.
[22] But signature schemes can always be derandomised by using a standard method, see [85].
[23] Also called a True Random Number Generator (TRNG).

(VM) environments, where repeated random bits may arise if they are extracted from the Operating System too soon after a VM image is reset [88].

There has been a long-running debate on whether such random bit generators should be *blocking* or *non-blocking*: if the OS keeps a running estimate of how much true entropy remains in the pool as output is consumed, then should the generator block further output being taken if the entropy estimate falls below a certain threshold? The short answer is no, if we believe we are using a cryptographically-secure PRNG to generate the output, provided the entropy pool is properly initialised with enough entropy after boot. This is because we should trust our PRNG to do a good job in generating output that is computationally indistinguishable from random, even if not truly random. Some modern operating systems now offer an interface to a random bit generator of this "non-blocking-if-properly-seeded" type.

# 3 KEY MANAGEMENT

[3, 55, 89, 90]

Cryptographic schemes shift the problem of securing data to that of securing and managing keys. Therefore no treatment of applied cryptography can ignore the topic of key management. As explained by Martin [3, Chapter 10], cryptographic keys are in the end just data, albeit of a special and particularly sensitive kind. So key management must necessarily involve all the usual processes involved in Information Security management, including technical controls, process controls and environmental controls.

An introductory treatment of key management can be found in the aforementioned [3, Chapter 10]. A more detailed approach can be found in the NIST three part series [55, 89, 90].

## 3.1 The Key Life-cycle

Keys should be regarded as having a life-cycle, from creation all the way to destruction.

Keys first need to be generated, which may require cryptographically secure sources of randomness, or even true random sources, in order to ensure keys have sufficient entropy to prevent enumeration attacks.

Keys may then need to be securely distributed to where they will be used. For example, a key may be generated as part of a smartcard personalisation process and injected into a smartcard from the personalisation management system through a physically secure channel; or a symmetric key for protecting a communication session may be established at a client and server in a complex cryptographic protocol, perhaps with one party choosing the session key and then making use of PKE to transport it to the other party.

Keys may also be derived from other keys using suitable cryptographic algorithms known as *Key Derivation Functions*.

Keys need to be stored securely until they are needed. We discuss some of the main key storage options in more detail in the sequel.

Then keys are actually used to protect data in some way. It may be necessary to impose limits on how much data the keys are used to protect, due to intrinsic limitations of the cryptographic scheme in which they are being used. Keys may then need to be changed or updated. For example, the TLS specification in its latest version, TLS 1.3 [91], contains recommendations about how much data each AEAD key in the protocol can be used to protect. These are set by analysing the security bounds for the employed AEAD schemes. TLS also features a key update sub-protocol enabling new keys to be established within a secure connection.

Keys may need to be revoked if they are discovered to have been compromised. The revocation status of keys must then be communicated to parties relying on those keys in a timely and reliable manner.

Keys may also need to be archived — put into long-term, secure storage — enabling the data they protect to be retrieved when needed. This may involve encrypting the keys under other keys, which themselves require management. Finally, keys should be securely deleted at the end of their lifetime. This may involve physical destruction of storage media, or carefully overwriting keys.

Given the complexity in the key life-cycle, it should be apparent that the key life-cycle and its attendant processes need to be carefully considered and documented as part of the design process for any system making use of cryptography.

We have already hinted that keys in general need to remain secret in order to be useful (public keys are an exception; as we discuss below, the requirement for public keys is that they be securely bound to identity of the key owner and their function). Keys can leak in many ways — through the key generation procedure due to poor randomness, whilst being transported to the place where they will be needed, through compromise of the storage system on which they reside, through side-channel attacks while in use, or because they are not properly deleted once exhausted. So it may be profitable for attackers to directly target keys and their management rather than the algorithms making use of them when trying to break a cryptographic system.

Additionally, it is good practice that keys come with what Martin [3] calls *assurance of purpose* — which party (or parties) can use the key, for which purposes and with what limits. Certain storage formats — for example, digital certificates — encode this information along with the keys. This relates to the *principle of key separation* which states that a given key should only ever be used for one purpose (or in one cryptographic algorithm). This principle is perhaps more often broken than observed and has led to vulnerabilities in deployed systems, see, for example [92, 56].

## 3.2 Key Derivation

*Key derivation* refers to the process of creating multiple keys from a single key. The main property required is that exposure of any of the derived keys should not compromise the security of any of the others, nor the root key from which they are derived. This is impossible in an information theoretic sense, since given enough derived keys, the root key must be determinable through an exhaustive search. But it can be assured under suitable computational assumptions. For example, suppose we have a Pseudo-Random Function (PRF) $F$ which takes as input key $K$ and "message" input $m$; then outputs $F(K, m_i)$ on distinct inputs $m_1, m_2, \ldots$ will appear to be random values to an adversary and even giving the adversary many input/output pairs $(m_i, F(K, m_i))$ will not help it in determining $K$ nor any further input/output pairs. Thus a pseudo-random function can be securely used as a Key Derivation Function (KDF) with root key $K$. We refer then to the $m_i$ as *labels* for key derivation.

In many situations, the root key $K$ may itself not be of a suitable length for use in a PRF or come from some non-uniform distribution. For example, the key may be a group element resulting from a Diffie-Hellman key exchange. In this case, one should first apply an entropy extraction step to make a suitable key $K$, then apply a PRF. One may also desire a key derivation function with variable length output (different functions may require keys of different sizes) or variable size label inputs. So the general requirements on a KDF go beyond what a simple PRF can offer. HKDF is one general-purpose KDF that uses the HMAC algorithm as a variable-input PRF. It is defined in [93]. As well as key and label inputs and variable length output, it features an optional non-secret *salt* input which strengthens security across multiple, independent invocations of the algorithm. In legacy or constrained systems, one can find many *ad hoc* constructions for KDFs, using for example block ciphers or hash functions.

Using a KDF allows for key diversification and makes it easier to comply with the principle of key separation. For example, one can use a single symmetric key to derive separate encryption and MAC keys to be used in an EtM construction. Modern AE schemes avoid this need, effectively performing key derivation internally. In the extreme, one might use a KDF in combination with a specific label to derive a fresh key for each and every application of a cryptographic algorithm, a process known in the financial cryptography context as *unique key per transaction*. One can also choose to derive further keys from a derived key, creating a *key hierarchy* or *tree of keys* of arbitrary (but usually bounded) depth. Such key hierarchies are commonly seen in banking systems. For example, a bank may have a payment-system-wide master secret, from which individual card secrets are derived; in turn, per transaction keys are derived from the card secrets. Of course, in such a system, protection of the master secret — the key to the kingdom! — is paramount. Generally specialised hardware is used for storing and operating with such keys.

## 3.3 Password-Based Key Derivation

A very common practice, arising from the inevitable involvement of humans in cryptographic systems, is to derive cryptographic keys from passwords (or passphrases). Humans cannot remember passwords that have the high entropy required to prevent exhaustive searches, so special-purpose KDFs should be used in such applications, called password-based KDFs (PBKDFs). The idea of these is to deliberately slow-down the KDF to limit the speed at which exhaustive searches can be carried out by an attacker (standard KDFs do not need to be slowed in this way, since they should only operate on high-entropy inputs). Memory-hard PBKDFs such as scrypt (defined in [94] and analysed in [95]) or Argon2 (the winner of a password hashing design competition[24]), which are designed to force an attacker to expend significant memory resources when carrying out exhaustive searches, should be used in preference to older designs. These PBKDFs have adjustable hardness parameters and permit salting, important for preventing pre-computation attacks on the collections of hashed passwords that are typically stored in authentication databases. It is worth keeping in mind that commercial password cracking services exist. These cater for many different formats and make extensive use of Graphical Processing Units (GPUs). They are impressively fast, rendering human-memorable passwords on the verge of being obsolete and forcing the adoption of either very long passwords or entirely different authentication methods.

An alternative to using a password directly to derive a key is to use the password as an authentication mechanism in a key exchange protocol, leading to the concept of Password Authenticated Key Exchange (PAKE). When designed well, a PAKE can limit an attacker to making a single password guess in each execution of the protocol. PAKE has not seen widespread adoption yet, but is currently undergoing standardisation in the IRTF.[25].

[24]See **https://www.password-hashing.net/**.
[25]Details at **https://github.com/cfrg/pake-selection**.

## 3.4 Key Generation

The main requirement for symmetric keys that are being generated from scratch is that they should be chosen close to uniformly at random from the set of all bit-strings of the appropriate key length. This requires access to good sources of random bits at the time when the key is selected. These bits may come from true random sources (e.g. from noise in electronic circuits or from quantum effects) or from an operating-system supplied source of pseudo-random bits, which may in turn be seeded and refreshed using random bits gathered from the local environment. This topic is discussed in more detail in Section 2. An alternative is to use a Physically Unclonable Function (PUF) to generate keying material from the intrinsic properties of a piece of hardware. Depending on what properties are measured, significant post-processing may be needed to correct for errors and to produce output with good randomness properties. An overview of some of the challenges arising for PUFs can be found in [96].

For asymmetric algorithms there may be additional requirements, for example the key pairs may require special algebraic structure or need to lie in certain numerical ranges. However, the `KeyGen` algorithms for such schemes should handle this internally and themselves only require access to a standard random bit source.

There are (in)famous cases where key generation processes were not appropriately randomised [86, 97] (see also the Debian incident[26]). Another challenge is that in some cases keys need to be generated in constrained environments, e.g. on smartcards that will be used as personal identity cards, where generating the keys “off-card” and then injecting them into the card would not meet the security requirements of the application. There may then be a temptation to over-optimise the key generation process. This can result in significant security vulnerabilities [98].

## 3.5 Key Storage

Once keys are generated, they typically need to be stored. An exception is where the keys can be generated on the fly from a human-memorable password, as discussed in Section 3.3.)

In general, the storage medium needs to be appropriately secured to prevent the keys becoming available to adversaries. In many cases, the only security for stored keys comes from that provided by the local operating system's access control mechanisms, coupled with hardware-enforced memory partitioning for different processes. Such security mechanisms can be bypassed by *local* attackers using low-level means that exploit the presence of shared resources between different executing processes, e.g. a shared memory cache. These are discussed in Section 2. This is a particular issue in multi-tenant computing environments, e.g. cloud computing, where a customer has little control over the processes running on the same CPU) as its own. But it is also an issue in single-tenant computing environments when one considers the possibility of malicious third-party software. An acute challenge arises for “crypto in the browser”, where users must rely on the browser to enforce separation and non-interference between code running in different browser tabs, and where the code running in one tab may be loaded from a malicious website “`evil.example.com`” that is trying to extract cryptographic secrets from code running another tab “`your-bank.example.com`”.

The Heartbleed incident [99] was so damaging precisely because it enabled a fully *remote* attacker (i.e. one not running on the same machine as the victim process but merely able

[26]See **https://www.debian.org/security/2008/dsa-1571**.

to connect over a network to that machine) to read out portions of a TLS server's memory, potentially including the server's private key. It shows that any amount of strong cryptography can easily be undermined thanks to a simple software vulnerability. In the case of Heartbleed, this was a buffer over-read in the OpenSSL implementation of the TLS/DTLS Heartbeat protocol.

Developers who store cryptographic keys in memory (or in software) often use software obfuscation techniques to try to make it harder to identify and extract the keys. Correspondingly, there are de-obfuscation tools which try to reverse this process and there is a long-running arms race in this domain. The research topic of white-box cryptography [100] attempts to formalise this game of attack and defence.

The alternative to in memory storage of keys is to rely on special purpose secure storage for keys. Devices for this exist at all scales and price points. For example, a smartcard may cost a few cents to manufacture, but can be well-protected against passive and invasive attacks that attempt to recover the keys that it stores. A well-designed smartcard will have an interface which carefully limits how an external card reader communicating with the card can interact with it. In particular, there will not be any commands that a reader can send to the card that would allow its keys to directly leave the card. Rather, a reader will only be able to send data to the card and have it cryptographically transformed locally using the stored keys, then receive the results of the cryptographic computations. Thus, the reader will be able to interact with the secrets on the card only via a cryptographic API. Smartcards are typically bandwidth- and computation-limited.

At the other end of the scale are Hardware Security Modules (HSMs), which may cost many thousands of US dollars, offer much greater capabilities and be tested to a high level of security using industry-standard evaluation methodologies (e.g. the NIST FIPS 140 series). HSMs are frequently used in the financial sector, and are also now offered in cloud environments, see Section 4. As with smartcards, an HSM offers key storage and key usage functions via a carefully controlled API. The API provided by an HSM can be used to extend the security that the HSM offers to keys that it directly stores to a larger collection of keys. Consider the simple case of using the HSM to store a Key Encryption Key (KEK), such that the HSM's API allows that key to be used internally for authorised encryption and decryption functions. Then the HSM can be used to wrap and, when needed, unwrap many Data Encryption Keys (DEKs) using a single KEK that is stored inside the HSM. Here wrapping means encrypting and unwrapping means decrypting. Assuming the used encryption mechanism is strong, the wrapped DEKs can be stored in general, unprotected memory. Further details on HSMs can be found in the Hardware Security CyBOK Knowledge Area [12].

TPMs also provide hardware-backed key storage. Technologies aiming to provide secure execution environments, such as Intel SGX and ARM Trustzone, enable secure storage of keys but also possess much more general capabilities. On mobile devices, the Android and iOS operating systems offer similar key storage features through Android Keystore and iOS Secure Enclave — both of these are essentially mini-HSMs.

## 3.6 Key Transportation

Depending on system design, keys may be generated at one place but need to be transported to another place where they will subsequently be used. Traditionally, secure couriers were used to physically transport keys stored on paper tape or magnetic media. Of course, this is costly and time-consuming. A related method, used often in low-value consumer applications (e.g. domestic wireless routers), is to simply print the key on the back of the device. Then security is reduced to that of the physical security of the device.

An alternative approach, widely used in the mobile telecommunications and consumer finance sector, is to inject symmetric keys into low-cost security modules (e.g. a Subscriber Identity Module, SIM, card in the case of mobile telephones or a bank card in the case of finance) and distribute those to customers. At the same time, copies of the symmetric keys may be kept *en masse* at a centralised location. In the case of mobile telecommunications, this is at a logical network component called the Authentication Centre (AuC); the symmetric keys are then used to perform authentication of SIM cards to the network and as the basis for deriving session keys for encrypting voice and data on the wireless portion of the network. In the financial setting, the symmetric keys injected into cards are typically already derived from master keys using a KDF, with the master keys being held in an HSM.

If a key is already in place, then that key can be used to transport fresh keys. Here, the idea is to sparingly use an expensive but very secure algorithm to transport the keys. With the advent of Public Key Cryptography, the problem of key transportation can be reduced to the problem of establishing an authentic copy of the public key of the receiver at the sender. As we discuss in Section 3.8, this is still a significant problem. Earlier versions of the TLS protocol used precisely this mechanism (with RSA encryption using PKCS#1 v1.5 padding) to securely transport a master key from a client in possession of a server's public key to that server. In TLS, various session keys are derived from this master key (using an *ad hoc* KDF construction based on iteration of MD5 and SHA-1).

## 3.7 Refreshing Keys and Forward Security

Consider a scenario where a symmetric key is being used to protect communications between two parties $A$ and $B$, e.g. by using the key in an AEAD scheme as was described in Section 1.18. Suppose that key is compromised by some attack — for example, a side-channel attack on the AEAD scheme. Then the security of *all* the data encrypted under that key is potentially lost. This motivates the idea of regularly refreshing keys. Another reason to do this, beyond key compromise, is that the formal security analysis of the AEAD scheme will indicate that any given key can only be used to encrypt a certain amount of data before the security guarantees are no longer meaningful.

Several mechanisms exist to refresh symmetric keys. A simple technique is to derive, using a KDF, a new symmetric key from the existing symmetric key at regular intervals, thereby creating a chain of keys. The idea is that if the key is compromised at some point in time, then it should still be hard to recover all previous keys (but of course possible to compute forward using the KDF to find all future keys). This property, where the security of old keys remains after the compromise of a current key, is somewhat perversely known as *forward security*. In our example it follows from standard security properties of a KDF. In practice, hashing is often improperly used in place of a KDF, leading to the notion of a *hash chain*. This approach does not require direct interaction between the different users of a given symmetric key (i.e.

$A$ and $B$ in our example). But it does need a form of synchronisation so that the two parties know which version of the key to use.

Another idea is to use a PKE scheme (more properly a KEM) to regularly transport a fresh symmetric key from one party to the other, e.g. from $A$ to $B$ under the protection of $B$'s public key and then use that *session* key in the AEAD scheme (as discussed above was done in earlier versions of TLS). This creates fresh session keys whenever needed. Now to fully evaluate the system under key compromises, we should consider the effect of compromise of $B$'s PKE private key too. This takes us back to square one: if the attacker can intercept the PKE ciphertexts as they are sent, store them and later learn $B$'s PKE private key by some means (e.g. by factorising the modulus in the case of RSA, or simply as a result of an order made by a lawful authority), then it can recover all the session keys that were ever transported and thereby decrypt all the AEAD ciphertexts. In short, if the capabilities of the adversary include PKE private key compromise, then the use of PKE does not achieve forward security.

To improve the situation, we can make use of *ephemeral* Diffie-Hellman key exchange, in combination with appropriate authentication to prevent active MiTM attacks, to set up fresh session keys. Here, *ephemeral* refers to the Diffie-Hellman values used being freshly generated by both parties in each key exchange instance. We might use digital signatures for the authentication, as TLS 1.3 does. Now what happens in the event of compromise of the equivalent of the KEM private key? This is the signing key of the digital signature scheme. We see, informally, that an active MiTM attacker who knows the signing key could spoof the authentication and fool the honest protocol participants into agreeing on fresh session keys with the attacker rather than each other. This cannot be prevented. However, a compromise at some point in time of the signing key has no effect on the security of previously established, old session keys. If the authentication was sound at the time of the exchange, so that active attacks were not possible, then an adversary has to break a Diffie-Hellman key exchange in order to learn a previously established session key. Moreover breaking one Diffie-Hellman key exchange does not affect the security of the others.[27] In comparison to the use of PKE to transport session keys, Diffie-Hellman key exchange achieves strictly stronger forward security properties.

Complementing forward security is the notion of backward security, aka post compromise, security [101]. This refers to the security of keys established *after* a key compromise has occurred. Using Diffie-Hellman key exchange can help here too, to establish fresh session keys, but only in the event that the adversary is restricted to being passive for at least one run of the Diffie-Hellman protocol.

[27] Except in the case where the discrete logarithm algorithm used allows reuse of computation when solving multiple DLPs. This is the case for the best known finite field algorithms, see [57] for the significant real-world security issues this can introduce, but not for the elliptic curve setting.

## 3.8 Managing Public Keys and Public Key Infrastructure

The main security requirement for a public key is that it be authentically bound to the identity of the party who is in legitimate possession of the corresponding private key. Otherwise impersonation attacks become possible. For example, a party $A$ might encrypt a session key to what it thinks is the public key of $B$, but the session key can be recovered by some other party $C$. Or a party $A$ might receive a digital signature on some message $M$ purporting to be generated by $B$ using its private key, but where the signature was actually generated by $C$; here $A$ would verify the signature using $C$'s (public) verification key thinking it was using $B$'s key.

A second requirement is that parties who rely on the soundness of the binding between public key and identity are also able to gain assurance that the binding is still valid (i.e. has not expired or been revoked).

These two requirements have many implications. Meeting them leads to the introduction of additional infrastructure and management functions. Conventionally, this collection of components is called a *Public Key Infrastructure (PKI)*.

### 3.8.1 Binding Public Keys and Identities via Certificates

A suitable mechanism to bind public keys and identities — and possibly other information — is needed. In early proposals for deploying Public Key Cryptography, it was proposed that this could take the form of a trusted bulletin board, where all the public keys and corresponding identities are simply listed. But this of course requires trust in the provider of the bulletin board service.

A non-scalable and inflexible solution, but one that is commonly used in mobile applications and IoT deployments, is to hard-code the required public key into the software of the party that needs to use the public key. Here, security rests on the inability of an adversary to change the public key by over-writing it in a local copy of the software, substituting it during a software update, changing it in the code repository of the software provider, or by other means.

Another solution is to use *public key digital certificates* (or just certificates for short). These are data objects in which the data includes identity, public key, algorithm type, issuance and expiry dates, key usage restrictions and potentially other fields. In addition, the certificate contains a digital signature, over all the other fields, of some Trusted Third Party (TTP) who attests to the correctness of the information. This TTP is known as a Certification Authority (CA). The most commonly used format for digital certificates is X.509 version 3 [102].

The use of certificates moves the problem of verifying the binding implied by the digital signature in the certificate to the authentic distribution of the CA's public key. In practice, the problem may be deferred several times via a *certificate chain*, with each TTP's public key in the chain being attested to via a certificate issued by a higher authority. Ultimately, this chain is terminated at the highest level by a root certificate that is self-signed by a root CA. That is, the root certificate contains the public verification key of the root CA and a signature that is created using the matching private signing key. A party wishing to make use of a user-level public key (called a relying party) must now verify a chain of certificates back to the root and also have means of assuring that the root public key is valid. This last step is usually solved by an out of band distribution of the root CA's public key. Root CAs may also cross-sign each other's root certificates.

As an important and visible example of a PKI, consider the *Web PKI*. Web browser vendors embed a list of the public keys of a few hundred different root CAs in their software and update the list from time to time via their software update mechanisms, which in turn may rely for its security on a separate PKI. Website owners pay to obtain certificates binding their sites' URLs to their public keys from subordinate CAs. Then, when running the TLS protocol for secure communications between a web browser and a website, the website's server sends a certificate chain to the web browser client. The chain provides the web browser with a copy of the server's public key (in the lowest certificate from the chain, the leaf or end-entity certificate) as well as a means of verifying the binding between the web site name in the form of its URL, and that public key. The operations and conventions of the Web PKI are managed by the CA/Browser Forum.[28]

### 3.8.2 Reliance on Naming, CA Operations and Time

In addition to needing a suitable binding mechanism, there must be a stable, controlled naming mechanism for parties. Moreover, parties need to have means of proving to CAs that they own a specific identity and CAs need to check such assertions. Equally, CAs need to be trusted to only issue certificates to the correct parties. This aspect of PKI intersects heavily with legal and regulatory aspects of Information Security and is covered in more detail in Law & Regulation CyBOK Knowledge Area [9].

For the Web PKI, there have been numerous incidents where CAs were found to have mis-issued certificates, either because they were hacked (e.g. DigiNotar[29]), because of poor control over the issuance process (e.g. TurkTrust[30]), or because they were under the control of governments who wished to gain surveillance capabilities over their citizens. This can lead to significant commercial impacts for affected CAs: in DigiNotar's case, the company went bankrupt. In other cases, CAs were found to not be properly protecting their private signing keys, leaving them vulnerable to hacking.[31] In response to a growing number of such incidents, Google launched the *Certificate Transparency (CT)* effort. CT provides an open framework for monitoring and auditing certificates; it makes use of multiple, independent public logs in an attempt to record all the certificates issued by browser-trusted CAs. The protocols and data formats underlying CT are specified in [103].[32]

Relying parties (i.e. parties verifying certificates and then using the embedded public keys) need access to reliable time sources to be sure that the certificate's lifetime, as encoded in the certificate, is still valid. Otherwise, an attacker could send an expired certificate for which it has compromised the corresponding private key to a relying party and get the relying party to use the certificate's public key. This requirement can be difficult to fulfill in low-cost or constrained environments, e.g. IoT applications.

[28] See **https://cabforum.org/**.
[29] See **https://en.wikipedia.org/wiki/DigiNotar**.
[30] See **https://nakedsecurity.sophos.com/2013/01/08/the-turktrust-ssl-certificate-fiasco-what-happened-and-what-happens-next/**.
[31] See for example the case of CNNIC, **https://techcrunch.com/2015/04/01/google-cnnic/**.
[32] See also **https://certificate.transparency.dev/** for the project homepage.

### 3.8.3 Reliance on Certificate Status Information

Relying parties verifying certificates also need access to reliable, timely sources of information about the status of certificates — whether the certificate is still valid or has been revoked for some security or operational reason. This can be done by regularly sending lists of revoked certificates to relying parties (known as Certificate Revocation Lists, CRLs), or having the relying parties perform a real-time status check with the issuing CA before using the public key using the Online Certificate Status Protocol, OCSP [104]. The former approach is more private for relying parties, since the check can be done locally, but implies the existence of a window of exposure for relying parties between the time of revocation and the time of CRL distribution. The latter approach provides more timely information but implies that large CAs issuing many certificates need to provide significant bandwidth and computation to serve the online requests.

In the web context, OCSP has become the dominant method for checking revocation status of certificates. OCSP's bandwidth issue is ameliorated by the practice of *OCSP stapling*, wherein a web server providing a certificate regularly performs its own OCSP check and includes the certified response from its CA along with its certificate. In an effort to further improve user privacy, in 2020, Mozilla experimentally deployed[33] an approach called CRLite developed in [105] in their Firefox browser. CRLite uses CT logs and other sources of information to create timely and compact CRLs for regular distribution to web browsers.

### 3.8.4 Reliance on Correct Software and Unbroken Cryptography

The software at a relying party that validates certificate chains needs to work properly. This is non-trivial, given the complexity of the X.509 data structures involved, the use of complex encoding languages and the need to accurately translate security policy into running code. There have been numerous failures. A prominent and entirely avoidable example is Apple's "goto fail" from 2014. Here a repeated line of code[34] for error handling in Apple's certificate verification code in its SSL/TLS implementation caused all certificate checking to be bypassed. This made it trivial to spoof a web server's public key in a fake certificate to clients running Apple's code. This resulted in a total bypass of the server authentication in Apple's SSL/TLS implementation, undermining all security guarantees of the protocol.[35]

The certificate industry has been slow to react to advances in the cryptanalysis of algorithms and slow to add support for new signature schemes. The story of SHA-1 and its gradual removal from the Web PKI is a prime example. This relates to the discussion of cryptographic agility in Section 1.14. The first cracks in SHA-1 appeared in 2005 [106]. Already at this point, cryptographers taking their standard conservative approach, recommended that SHA-1 be deprecated in applications requiring collision resistance. From 2005 onwards, the cryptanalysis of SHA-1 was refined and improved. Finally in 2017, the first public collisions for SHA-1 were exhibited [107]. This was followed in 2019 by a chosen-prefix collision attack that directly threatened the application of SHA-1 in certificates [108]. However, despite the direction of travel having been clear for more than a decade, it took until 2017 before the major web browsers finally stopped accepting SHA-1 in Web PKI certificates. Today, SHA-1 certificates are still to be found in payment systems and elsewhere. The organisations running these systems are inherently change-averse because they have to manage complex systems that must

[33] See **https://blog.mozilla.org/security/2020/01/09/crlite-part-1-all-web-pki-revocations-compressed/**.
[34] The offending line of code was literally "`goto fail`".
[35] See **https://dwheeler.com/essays/apple-goto-fail.html** for a detailed write-up of the incident and its implications for Apple's software development processes.

continue to work across algorithm and technology changes. In short, these organisations are not cryptographically agile, as discussed in Section 1.14.

### 3.8.5 Other Approaches to Managing Public Keys

The *web of trust* is an alternative to hierarchical PKIs in which the users in a system vouch for the authenticity of one another's public keys by essentially cross-certifying each other's keys. It was once popular in the PGP community but did not catch on elsewhere. Such a system poses significant usability challenges for ordinary users [109].

*Identity-Based Cryptography (IBC)* [110] offers a technically appealing alternative to traditional Public Key Cryptography in which users' private keys are derived directly from their identities by a TTP called the Trusted Authority (TA) in possession of a master private key. The benefit is that there is no need to distribute public keys; a relying party now needs only to know an identity and have an authentic copy of the TA's public key. The down-side for many application domains is that trust in the TA is paramount, since it has the capability to forge users' signatures and decrypt ciphertexts intended for them through holding the master private key. On the other hand, IBC's built-in key escrow property may be useful in corporate security applications. *Certificateless Cryptography* [111] tries to strike a balance between traditional PKI and IBC. These and other related concepts have sparked a lot of scientific endeavour, but little deployment to date.

# 4 CONSUMING CRYPTOGRAPHY

The content of this section is based on the author's personal experience.

## 4.1 The Challenges of Consuming Cryptography

Numerous people, including those who are well educated in computer science and mathematics, can and do regularly make fundamental errors when attempting to devise novel cryptographic schemes or to implement existing mechanisms.

Perhaps one reason for this is that many people receive informal exposure to cryptography through popular culture, where it is often shown in a simplified way, for example using pen and paper ciphers or watching a password being revealed character-by-character through some unspecified search process. The subject also has basic psychological appeal — after all, who does not love receiving secret messages?

The issue shows up in several different ways. Least dangerous, the author has seen that cryptographic conferences and journals regularly receive submissions from people who are not aware of the advanced state-of-the-art. A classic trope is papers on encryption algorithms for digital images, where usually some low-level manipulation of pixel data is involved and security rests on taking a standard benchmark picture from the image processing community and showing that it is visually scrambled by the algorithm. (Of course, image data is ultimately represented by bits and standard cryptographic algorithms operate on those bits.) Other topics common in such papers are chaos-based cryptography and combining multiple schemes (RSA, ElGamal, etc) to make a stronger one. The activity of generating such papers is a waste of time for the authors and reviewers alike, while it misleads students involved in writing the papers about the true nature of cryptography as a research topic.

This author has seen multiple examples where complete outsiders to the field have been persuaded to invest in cryptographic technologies which either defy the laws of information theory or which fall to the "kitchen sink" fallacy of cryptographic design — push the data through enough complicated steps and it must be secure. Another classic design error is for an inventor to fall under the spell of the "large keys" fallacy: if an algorithm has a very large key space, then surely it must be secure? Certainly a large enough key space is necessary for security, but it is far from sufficient. A third fallacy is that of "friendly cryptanalysis": the inventor has tried to break the new algorithm themselves, so it must be secure. There is no substitute for independent analysis.

Usually these technologies are invented by outsiders to the field. They may have received encouragement from someone who is a consumer of cryptography but not themselves an expert or someone who is too polite to deliver a merciful blow. Significant effort may be required to dissuade the original inventors and their backers from taking the technology further. A sometimes useful argument to deploy in such cases is that, while the inventor's idea may or may not be secure, we already have available standardised, carefully-vetted, widely-deployed, low-cost solutions to the problem and so it will be hard to commercialise the invention in a heavily commoditised area.

Another set of issues arise when software developers, perhaps with the best of intentions and under release deadline pressure, "roll their own crypto". Maybe having taken an introductory course in Information Security or Cryptography at Bachelor's level, they have accrued enough knowledge not to try to make their own low-level algorithms and they know they can use an API to a cryptographic library to get access to basic encryption and signing functions. However, with today's cryptographic libraries, it is easy to accidentally misuse the API and end up with an insecure system. Likely the developer wants to do something more complex than simply encrypting some plaintext data, but instead needs to plug together a collection of cryptographic primitives to do something more complex. This can lead to the "kitchen sink" fallacy at the system level. Then there is the question of how the developer's code should deal with key management — recall that cryptographic schemes only shift the problem of securing data to that of securing keys. Unfortunately, key management is rarely taught to Bachelor's students as a first class issue and this author has seen that basic issues like hard-coded keys are still found on a regular basis in deployed cryptographic software.

## 4.2 Addressing the Challenges

How can individuals and businesses acting as consumers of cryptography avoid these problems? Some general advice follows.

There is no free lunch in cryptography. If someone without a track record or proper credentials comes bearing a new technological breakthrough, or just a new algorithm: beware. Consumers can look for independent analyses by reputable parties — there are companies and individuals who can provide meaningful cryptographic review. Any reputable consultant can supply a list of recent consulting engagements and contact details for references. Consumers can also check for scientific publications in reputable academic venues that support any new technology; very few crank ideas survive peer review, and consumers should look for clear indications of publication quality. Consumers can learn to detect cryptographic snake-oil, for example, by looking for instances of the kitchen sink, large keys and friendly cryptanalysis fallacies.[36]

[36] A more extensive list of snake-oil indicators can be found at: http://www.interhack.net/people/cmcurtin/

Developers should not roll their own cryptographic algorithms. They should rely on vetted, standardised algorithms already available in packaged forms through cryptographic libraries. Ideally, they should not roll their own higher-level cryptographic systems and protocols, but rely on existing design patterns and standards. As just one example, there is usually no need to build a new, secure, application-layer communications protocol when SSL/TLS support is ubiquitous. Developers who are developing cloud-based solutions (an extremely common use-case) should make use of cryptographic services from cloud providers — an emerging approach called "Cryptography-as-a-Service" (CaaS). This is a natural extension of the "Software-as-a-Service" paradigm, but commonly extends it to include key management services. A key feature in commercial CaaS offerings is "HSM-as-a-service", allowing service users to avoid the cost and expertise needed to maintain on-premise HSMs.[37]

When these options are not possible, developers should seek expert advice (and not Crypto Stack Exchange [112]). Very large organisations should have an in-house cryptography development team who vet all uses of cryptography before they go into production, ideally with involvement from the design stage. There is at least one large software company that today runs detection tools across its entire codebase to find instances where cryptography is being misused, and sends email alerts to the cryptography development team.

Smaller organisations for whom cryptography is a core technology should employ applied cryptographers with proven credentials in the field. Alternatively such medium-sized organisations — and the smallest companies — should cultivate relationships with trusted external partners who can provide cryptographic consulting services. The cost of such services is relatively small compared to the potential damage to reputation and shareholder value in the event of a major security incident arising from the improper use of cryptography.

## 4.3 Making Cryptography Invisible

Ultimately, the best kind of cryptography is that which is as invisible as possible to its end users. Five years ago web browsers displayed locks of different colours to indicate whether SSL/TLS connections were secure or not, but users could always click-through to reach the website regardless. Today, web browsers like Google Chrome and Mozilla Firefox are more aggressive in how they handle SSL/TLS security failures and in some cases simply do not allow a connection to be made to the website.[38] There is a continuous tension here between protecting users and enabling functionality that users demand. For example, at the time of writing, users can still freely visit websites that do not offer SSL/TLS connections in both Chrome and Firefox, receiving a "not secure" warning in the browser bar; this behaviour may change in the future as more and more websites switch to supporting SSL/TLS.

Meanwhile, for more than 25 years, in most jurisdictions, mobile telephone calls have been encrypted between the mobile device and the base-station (or beyond) in order to prevent eavesdropping on the broadcast communications medium. However, even in the latest 5G systems, the encryption is optional [113, Annex D] and can be switched off by the network operator. However, very few mobile telephones actually display any information to the user

**snake-oil-faq.html**.

[37]Without wishing to favour any particular vendors, interested readers can learn more about the typical features of such services at **https://aws.amazon.com/cloudhsm/** or **https://www.entrust.com/digital-security/certificate-solutions/products/pki/managed-services/entrust-cryptography-as-a-service**.

[38]Readers interested in learning more about what different browsers do and do not allow can visit **https://badssl.com/** or **http://example.com**.

about whether encryption is enabled or not, so it is debatable to what extent users are really protected.

By contrast, secure messaging services like Signal offer end-to-end security by default and with no possibility of turning these security features off. Signal's servers are not able to read users' messages — as a January 2021 advert for Signal pointedly stated, "we know nothing". Cryptographic security and user privacy are core to the company's business model. So much so that if a significant cryptographic flaw were to be found in Signal's design, then, at the very least, it would lead to a significant loss of customers and possibly even lead to the company's downfall. Yet there is very little in-app security messaging — there is no equivalent of a browser lock that users are asked to interpret, for example.

# 5 APPLIED CRYPTOGRAPHY IN ACTION

[91, 114, 115]

Having explored the landscape of applied cryptography from the bottom up, we now take a brief look at three different application areas, using them to draw out some of the key themes of the KA.

## 5.1 Transport Layer Security

The Transport Layer Security (TLS) protocol has already been mentioned several times in passing. The protocol has a long history, arriving at TLS 1.3 [91], completed in 2018. It is a complex protocol with several interrelated sub-protocols: the TLS Handshake Protocol uses (usually) asymmetric cryptography to establish session keys; these are then used in the TLS Record Protocol in conjunction with symmetric techniques to provide confidentiality and integrity for streams of application data; meanwhile the TLS Alert Protocol can be used to transport management information and error alerts. TLS is now used to protect roughly 95% of all HTTP traffic.[39] In this application domain, TLS relies heavily on the Web PKI for server authentication.

TLS 1.3 represents a multi-year effort by a coalition of industry-based engineers and academics working under the aegis of the IETF to build a general-purpose secure communications protocol. The main drivers for starting work on TLS 1.3 were, firstly, to improve the security of the protocol by updating the basic design and removing outdated cryptographic primitives (e.g. switching the MtE construction for AEAD only; removing RSA key transport in favour of DHKE to improve forward security) and, secondly, to improve the performance of the protocol by reducing the number of communication round trips needed to establish a secure channel. Specifically, in earlier versions of TLS, two round trips were needed, while in TLS 1.3 only one is needed in most cases and even a zero round trip mode is supported in TLS 1.3 when keys have been established in a previous session.

The design process for TLS 1.3 is described and contrasted with the process followed for previous versions in [49]. A key difference is the extensive reliance on formal security analysis during the design process. This resulted in many research papers being published along the way, amongst which we highlight [116, 117, 118]. Some of the formal analysis was able to detect potential security flaws in earlier versions of the protocol, thereby influencing the

[39]See **https://transparencyreport.google.com/https/overview?hl=en** for Google data supporting this statistic.

protocol design in crucial ways — for example, the use of HKDF in a consistent, hierarchical manner to process keying material from different stages of the protocol. The protocol was also modified to make it more amenable to formal security analysis. It is an almost complete success story in how academia and industry can work together. Only a few criticisms can be levelled at the process and the final design:

- Not all of the security and functionality requirements were elicited at the outset, which meant many design changes along the way, with attendant challenges for researchers doing formal analysis. However, such an iterative approach seems to be unavoidable in a complex protocol designed to serve many use cases.
- The formal analyses missed a few corner cases, especially in the situation where authentication is based on pre-shared symmetric keys. An attack in one such corner case was subsequently discovered [119].
- The zero round trip mode of TLS 1.3 has attractive latency properties but achieves these at the expense of forward security. Defending against replay attacks in this mode is difficult in general and likely impossible in distributed server settings when interactions with typical web clients are taken into account. Recent research showing how to add forward security to the zero round trip mode of TLS 1.3 can be found in [120, 121, 122, 123].

## 5.2 Secure Messaging

On the surface, secure messaging seems to have a very similar set of requirements to TLS: we have pairs of communicating parties who wish to securely exchange messages. However, there are significant differences, leading to different cryptographic solutions. First, secure messaging systems are asynchronous, meaning that the communicating parties cannot easily run an equivalent of the TLS Handshake Protocol to establish symmetric keys. Second, there is no infrastructure analogous to the Web PKI that can be leveraged to provide authentication. Third, group communication, rather than simply pairwise communication, is an important use case. Fourth, there is usually a bespoke server that is used to relay messages between the pairs of communicating parties but which should not have access to the messages themselves.

We discuss three different secure messaging systems: Apple's iMessage, Signal (used in WhatsApp) and Telegram. We focus on the two-party case for brevity.

### 5.2.1 Apple iMessage

Apple's iMessage system historically used an *ad hoc* signcryption scheme that was shown to have significant vulnerabilities in [124]. This was despite signcryption being a well-known primitive with well-established models, generic constructions from PKE and digital signatures and security proofs in the academic literature. Conjecturally, the designers of Apple's scheme were constrained by the functions available in their cryptographic library. The Apple system relied fully on trust in Apple's servers to distribute authentic copies of users' public keys — a PKI by fiat. The system was designed to be end-to-end secure, meaning that without active impersonation via key substitution, Apple could not read users' messages. It did not enjoy any forward-security properties, however: once a user's private decryption key was known, all messages intended for that user could be read. Note that Apple's iMessage implementation is not open source; the above description is based on the reverse engineering carried out in [124] and so may no longer be accurate.

### 5.2.2 Signal

The Signal design, which is used in both Signal and WhatsApp, takes a slightly different approach in the two-party case. It uses a kind of asynchronous DHKE approach called *ratcheting*. At a high level, every time Alice sends user Bob a new message, she also includes a Diffie-Hellman (DH) value and updates her symmetric key to one derived from that DH value and the DH value she most recently received from Bob. On receipt, Bob combines the incoming DH value with the one he previously sent to make a new symmetric key on his side. This key is called a chaining key.

For each message that Alice sends to Bob without receiving a reply from Bob, she derives two new keys from the current chaining key by applying a KDF (based on HKDF) to it; one key is used as the next chaining key, the other is used to encrypt the current message. This is also called ratcheting by the Signal designers and the combination of ratcheting applied to both DH values and symmetric keys is called *double ratcheting*.[40] This mechanism provides forward security for Signal messages, despite its asynchronous nature. It also provides post compromise security. The use of ratcheting, however, entails problems with synchronisation: if a message is lost between Alice and Bob, then their keys will end up in different states. This is solved by keeping caches of recent chaining keys.

For symmetric encryption, Signal uses a simple generic AE construction based on EtM relying on CBC mode using AES with 256 bit keys for the “E” component and HMAC with SHA-256 for the “M” component. This is a conservative and well-understood design.

Authentication in Signal is ultimately the same as in iMessage: it depends on trust in the server. The idea is that users register a collection of DH values at the server; these are fetched by other users and used to establish initial chaining keys. However, a malicious server could replace these values and thereby mount a MitM attack. The use of human-readable key fingerprints provides mitigation against this attack.

A formal security analysis of the double ratcheting process used by Signal can be found in [114]. Note that, in order to tame complexity, this analysis does not treat the composition of the double ratchet with the symmetric encryption component. The Signal design has spurred a spate of recent research into the question of what is the best possible security one can achieve in two-party messaging protocols and how that security interacts with the synchronisation issues.

### 5.2.3 Telegram

A third design is that used by Telegram.[41] It is notable for the way it combines various cryptographic primitives (RSA, finite field DHKE, a hash-based key derivation function, a hash-based MAC and a non-standard encryption mode called IGE). Moreover, it does not have proper key separation: keys used to protect messages from Alice to Bob share many overlapping bits with keys used in the opposite direction; moreover those key bits are taken directly from “raw” DHKE values. These features present significant barriers to formal analysis and violate cryptographic best practices. Furthermore, Telegram does not universally feature end-to-end encryption; rather it has two modes, one of which is end-to-end secure, the other of which provides secure communications only from each client to the server. The latter seems to be much more commonly used in practice, but is of course subject to interception. This

[40] See **https://signal.org/docs/specifications/doubleratchet/** for a concise overview of the process.
[41] See **https://telegram.org/**.

is concerning, given that Telegram is frequently used by higher-risk users in undemocratic countries.

## 5.3 Contact Tracing à la DP-3T

The DP-3T project[42] was formed by a group of academic researchers in response to the COVID-19 pandemic, with the core aim of rapidly developing automated contact tracing technology based on mobile phones and Bluetooth Low Energy beacons. A central objective of DP-3T was to enable automated contact tracing in a privacy-preserving way, so without using location data and without storing lists of contacts at a central server. DP-3T's approach directly influenced the Google-Apple Exposure Notification (GAEN)[43] system that forms the basis for dozens of national contact tracing apps around the world, including (after a false start) the UK system. An overview of the DP-3T proposal is provided in [115].

The DP-3T design uses cryptography at its heart. Each phone generates a symmetric key and uses this as the root of a chain of keys, one key per day. Each day key in the chain is then used to generate, using a pseudo-random generator built from AES in CTR mode, a sequence of 96 short pseudo-random strings called *beacons*. At each 15-minute time interval during the day, the next beacon from the sequence is selected and broadcast using BLE. Other phones in the vicinity pick up and record the beacon-carrying BLE signals and store them in a log along with metadata (time of day, received signal strength). Notice that the beacons are indistinguishable from random strings, under the assumption that AES is a good block cipher.

When a user of the system receives a positive COVID-19 test, they instruct their phone to upload the recent day keys to a central server, possibly along with sent signal strength information.[44] All phones in the system regularly poll the server for the latest sets of day keys, use them to regenerate beacons and look in their local logs to test if at some point they came into range of a phone carried by a person later found to be infected. Using sent and received signal strength information in combination with the number and closeness (over time) of matching beacons, the phone can compute a risk score. If the score is above a threshold, then the phone user can be instructed to get a COVID-19 test themselves. Setting the threshold in practice to balance false positives against false negatives is a delicate exercise made more difficult by the fact that BLE permits only inaccurate range estimation.

Notice that the central server in DP-3T stores only day keys released from phones by infected parties. The central server is not capable of computing which users were in proximity to which other users, nor even the identity of users who uploaded keys (though this information will become visible to the health authority because of the issuance of authorisation codes). All the comparison of beacons and risk computations are carried out on users' phones. One can contend that a fully centralised system could provide more detailed epidemiological information — some epidemiologists unsurprisingly made this argument. On the other hand, the strict purpose of the DP-3T system was to enable automated contact tracing, not to provide an epidemiological research tool. A more detailed privacy analysis of DP-3T can be found in [115].

The DP-3T design was produced, analysed, prototyped and deployed under test conditions all in the space of a few weeks. After adoption and adaptation by Google and Apple, it made its

[42]https://github.com/DP-3T/documents

[43]See **https://www.google.com/covid19/exposurenotifications/** and **https://covid19.apple.com/contacttracing**.

[44]To prevent spurious uploads, the day keys can only be uploaded after entering an authorisation code issued by the local health authority into the app.

way into national contact tracing apps within a few months. Given the pace of development, simplicity of the core design was key. Only "off the shelf" cryptographic techniques available in standard cryptographic libraries could be used. Given the likely scaling properties of the system (possibly tens of millions of users per country) and the constraints of BLE message sizes, using Public Key Cryptography was not an option; only symmetric techniques could be countenanced. Many follow-up research papers have proposed enhanced designs using more complex cryptographic techniques. The DP-3T team did not have this luxury and instead stayed resolutely pragmatic in designing a system that balances privacy, functionality and ease of deployment, and that resists repurposing.

# 6 THE FUTURE OF APPLIED CRYPTOGRAPHY

The first 2000 years of applied cryptography were mostly about securing data in transit, starting from the Caesar cipher and ending with today's mass deployment of TLS, secure messaging and privacy-preserving contact tracing systems. Today, cryptography is also heavily used to protect data at rest, the second element in our cryptographic triumvirate from the KA's introduction. These two application domains will continue to develop and become more pervasive in our everyday lives — consider for example the rise of the Internet of Things and its inherent need for communications security.

We anticipate significant developments in the following directions:

- The debate around lawful access to encrypted data will continue. However, we reiterate that in the face of determined but not even particularly sophisticated users, this is a lost battle. For example, it seems unlikely that state security agencies can break the cryptography that is used in Signal today. Instead, they will have to continue to bypass cryptographic protections through exploitation of vulnerabilities in end systems. Legal frameworks to enable this exist in many countries. Of course, governments could pass legislation requiring developers to include mechanisms to enable lawful access, or could pressure vendors into removing secure messaging applications from app stores.
- We expect that the current focus on cryptocurrencies and blockchains will result in a core set of useful cryptographic technologies. For example, anonymous cryptocurrencies have already been an important vehicle for forcing innovation in and maturation of zero-knowledge proofs.
- The third element in our cryptographic triumvirate was cryptography for data under computation. This area is undergoing rapid technical development and there is a healthy bloom of start-up companies taking ideas like FHE and MPC to market. A good overview of the status for "applied MPC" can be found in [125], while [126] provides insights into deployment challenges specific to FHE. The idea of being able to securely outsource one's data to third party providers and allow them to perform computations on it (as FHE does) is very alluring. However, some 15 years after its invention, FHE still incurs something like a $10^6$—$10^8$ times overhead compared to computing on plaintext data. This limits its application to all but the most sensitive data at small scales. Meanwhile, the core applications become ever more data- and computation-hungry, so remain out of reach of FHE for now. FHE, MPC and related techniques also face a significant challenge from trusted execution technologies like SGX. Approaches like SGX still rely on cryptography for attesting to the correct execution of code in secure enclaves, but can effectively emulate FHE functionality without such large overheads. On the other hand,

SGX and related technologies have themselves been the subject of a series of security vulnerabilities so may offer less protection in practice than cryptographic approaches.

- One area where computing on encrypted data may have a medium-term commercial future is in specialised applications such as searching over encrypted data and more generally, database encryption. Current state-of-the-art solutions here trade efficiency (in terms of computation and bandwidth overheads) for a certain amount of leakage. Quantifying the amount of leakage and its impact on security is a challenging research problem that must be solved before these approaches can become widely adopted.[45]

- Another growth area for applied cryptography is in privacy-preserving techniques for data-mining and data aggregation. Google's privacy-preserving advertising framework [127] provides one prominent example. Another is the Prio system [128] that allows privacy-preserving collection of telemetry data from web browsers. Prio has been experimentally deployed in Mozilla's Firefox browser.[46]

- Electronic voting (e-voting) has long been touted as an application area for cryptography. There is a large scientific literature on the problem. However, the use of e-voting in local and national elections has proved problematic, with confidence-sapping security vulnerabilities having been found in voting software and hardware. For example, a recent Swiss attempt to develop e-voting was temporarily abandoned after severe flaws were found in some of the cryptographic protocols used in the system during a semi-open system audit [129]. The Estonian experience has been much more positive, with a system built on Estonia's electronic identity cards having been in regular use (and having seen regular upgrades) since 2005. Key aspects of the Estonian success are openness, usability and the population's broad acceptance of and comfort with online activity.

- We may see a shift in how cryptography gets researched, developed and then deployed. The traditional model is the long road from research to real-world use. Ideas like MPC have been travelling down this road for decades. Out of sheer necessity, systems like DP-3T have travelled down the road much more quickly. A second model arises when practice gets ahead of theory and new theory is eventually developed to analyse what is being done in practice; often this leads to a situation where the practice could be improved by following the new theory, but the improvements are slow in coming because of the drag of legacy code and the difficulty of upgrading systems in operation. Sometimes a good attack is needed to stimulate change. A third model is represented by TLS 1.3: academia and industry working together to develop a complex protocol over a period of years.

- Cryptography involves a particular style of thinking. It involves quantifying over all adversaries in security proofs (and not just considering particular adversarial strategies), being conservative in one's assumptions, and rejecting systems even if they only have "certificational flaws". Such adversarial thinking should be more broadly applied in security research. Attacks on machine learning systems are a good place where this cross-over is already bearing fruit.

[45]But see **https://docs.mongodb.com/manual/core/security-client-side-encryption/** for details of MongoDB's use of deterministic symmetric encryption to enable searchable field-level encryption.

[46]See **https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/**.

## CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

| Topics | Cites |
|---|---|
| 1 Algorithms, Schemes and Protocols | [2, 3, 4] |
| 2 Cryptographic Implementation | [25, 67, 68, 69] |
| 3 Key Management | [3, 55, 89, 90] |
| 4 Consuming Cryptography | |
| 5 Applied Cryptography in Action | [91, 114, 115] |
| 6 The Future of Applied Cryptography | |

## REFERENCES

[1] N. Smart, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Cryptography, version 1.0.1. [Online]. Available: https://www.cybok.org/

[2] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, 0.5 ed., 2020. [Online]. Available: http://toc.cryptobook.us/

[3] K. Martin, *Everyday Cryptography: Fundamental Principles and Applications*, 2nd ed. Oxford University Press, 2017.

[4] N. P. Smart, *Cryptography Made Simple*, ser. Information Security and Cryptography. Springer, 2016.

[5] M. R. Albrecht, J. Blasco, R. B. Jensen, and L. Mareková, "Collective information security in large-scale urban protests: the case of Hong Kong," in *30th USENIX Security Symposium (USENIX Security '21)*. USENIX Association, 2021.

[6] C. Troncoso, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Privacy & Online Rights, version 1.0.2. [Online]. Available: https://www.cybok.org/

[7] A. Greenberg, "Monero, the drug dealer's cryptocurrency of choice, is on fire," 2017. [Online]. Available: https://www.wired.com/2017/01/monero-drug-dealers-cryptocurrency-choice-fire/

[8] C. Dion-Schwarz, D. Manheim, and P. B. Johnston, "Terrorist use of cryptocurrencies: Technical and organizational barriers and future threats," 2019. [Online]. Available: https://www.rand.org/content/dam/rand/pubs/research_reports/RR3000/RR3026/RAND_RR3026.pdf

[9] R. Carolina, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Law & Regulation, version 1.0.2. [Online]. Available: https://www.cybok.org/

[10] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, M. Green, S. Landau, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier, M. A. Specter, and D. J. Weitzner, "Keys under doormats: mandating insecurity by requiring government access to all data and communications," *Journal of Cybersecurity*, vol. 1, no. 1, pp. 69–79, 11 2015. [Online]. Available: https://doi.org/10.1093/cybsec/tyv009

[11] K. Martin, *Cryptography: The Key to Digital Security, How It Works, and Why It Matters*. W. W. Norton & Company, 2020.

[12] I. Verbauwhede, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Hardware Security, version 1.0.1. [Online]. Available: https://www.cybok.org/

[13] NIST, "Secure Hash Standard (SHS)," August 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[14] ——, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,"

August 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[15] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. D. Jr., "Advanced Encryption Standard (AES)," November 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[16] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," November 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[17] J. Mason, K. Watkins, J. Eisner, and A. Stubblefield, "A natural language approach to automated cryptanalysis of two-time pads," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 2006, pp. 235–244. [Online]. Available: https://doi.org/10.1145/1180405.1180435

[18] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Internet Engineering Task Force, Feb. 1997, updated by RFC 6151. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt

[19] P. Rogaway, "Nonce-based symmetric encryption," in *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, ser. Lecture Notes in Computer Science, B. K. Roy and W. Meier, Eds., vol. 3017. Springer, 2004, pp. 348–359. [Online]. Available: https://doi.org/10.1007/978-3-540-25937-4_22

[20] H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic, "Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS," in *10th USENIX Workshop on Offensive Technologies, WOOT 16, Austin, TX, USA, August 8-9, 2016*, N. Silvanovich and P. Traynor, Eds. USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/bock

[21] S. Gueron, A. Langley, and Y. Lindell, "AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption," RFC 8452 (Informational), RFC Editor, Fremont, CA, USA, Apr. 2019. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8452.txt

[22] P. Rogaway, "Authenticated-encryption with associated-data," in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, V. Atluri, Ed. ACM, 2002, pp. 98–107. [Online]. Available: https://doi.org/10.1145/586110.586125

[23] P. Rogaway and T. Shrimpton, "A provable-security treatment of the key-wrap problem," in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer, 2006, pp. 373–390. [Online]. Available: https://doi.org/10.1007/11761679_23

[24] S. Vaudenay, "Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ..." in *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, ser. Lecture Notes in Computer Science, L. R. Knudsen, Ed., vol. 2332. Springer, 2002, pp. 534–546. [Online]. Available: https://doi.org/10.1007/3-540-46035-7_35

[25] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 526–540. [Online]. Available: https://doi.org/10.1109/SP.2013.42

[26] M. R. Albrecht, K. G. Paterson, and G. J. Watson, "Plaintext recovery attacks against

SSH," in *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. IEEE Computer Society, 2009, pp. 16–26. [Online]. Available: https://doi.org/10.1109/SP.2009.5

[27] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, ser. Lecture Notes in Computer Science, T. Okamoto, Ed., vol. 1976. Springer, 2000, pp. 531–545. [Online]. Available: https://doi.org/10.1007/3-540-44448-3_41

[28] C. Namprempre, P. Rogaway, and T. Shrimpton, "Reconsidering generic composition," in *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, ser. Lecture Notes in Computer Science, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Springer, 2014, pp. 257–274. [Online]. Available: https://doi.org/10.1007/978-3-642-55220-5_15

[29] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 8439 (Informational), RFC Editor, Fremont, CA, USA, Jun. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8439.txt

[30] M. Abdalla, M. Bellare, and G. Neven, "Robust encryption," *J. Cryptol.*, vol. 31, no. 2, pp. 307–350, 2018. [Online]. Available: https://doi.org/10.1007/s00145-017-9258-8

[31] K. Moriarty (Ed.), B. Kaliski, J. Jonsson, and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," RFC 8017 (Informational), RFC Editor, Fremont, CA, USA, Nov. 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8017.txt

[32] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," in *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, ser. Lecture Notes in Computer Science, H. Krawczyk, Ed., vol. 1462. Springer, 1998, pp. 1–12. [Online]. Available: https://doi.org/10.1007/BFb0055716

[33] H. Böck, J. Somorovsky, and C. Young, "Return of Bleichenbacher's oracle threat (ROBOT)," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 817–849. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[34] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption," *J. Cryptol.*, vol. 17, no. 2, pp. 81–104, 2004. [Online]. Available: https://doi.org/10.1007/s00145-002-0204-y

[35] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976. [Online]. Available: https://doi.org/10.1109/TIT.1976.1055638

[36] C. Boyd, A. Mathuria, and D. Stebila, *Protocols for Authentication and Key Establishment*, 2nd ed., ser. Information Security and Cryptography. Springer, 2020.

[37] M. R. Albrecht, J. Massimo, K. G. Paterson, and J. Somorovsky, "Prime and prejudice: Primality testing under adversarial conditions," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 281–298. [Online]. Available: https://doi.org/10.1145/3243734.3243787

[38] S. D. Galbraith, J. Massimo, and K. G. Paterson, "Safety in numbers: On the need for robust Diffie-Hellman parameter validation," in *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography,*

*Beijing, China, April 14-17, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, D. Lin and K. Sako, Eds., vol. 11443. Springer, 2019, pp. 379–407. [Online]. Available: https://doi.org/10.1007/978-3-030-17259-6_13

[39] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985. [Online]. Available: https://doi.org/10.1109/TIT.1985.1057074

[40] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle Diffie-Hellman assumptions and an analysis of DHIES," in *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, ser. Lecture Notes in Computer Science, D. Naccache, Ed., vol. 2020. Springer, 2001, pp. 143–158. [Online]. Available: https://doi.org/10.1007/3-540-45353-9_12

[41] A. Menezes and N. P. Smart, "Security of signature schemes in a multi-user setting," *Des. Codes Cryptogr.*, vol. 33, no. 3, pp. 261–274, 2004. [Online]. Available: https://doi.org/10.1023/B:DESI.0000036250.18062.3f

[42] M. Bellare and P. Rogaway, "The exact security of digital signatures - how to sign with RSA and Rabin," in *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Springer, 1996, pp. 399–416. [Online]. Available: https://doi.org/10.1007/3-540-68339-9_34

[43] NIST, "Digital Signature Standard (DSS)," July 2013. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[44] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032 (Informational), RFC Editor, Fremont, CA, USA, Jan. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8032.txt

[45] L. Lamport, "Constructing digital signatures from a one-way function," SRI International Computer Science Laboratory, Tech. Rep. SRI-CSL-98, October 1979.

[46] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Plenum Press, New York, 1982, pp. 199–203. [Online]. Available: https://doi.org/10.1007/978-1-4757-0602-4_18

[47] E. F. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, V. Atluri, B. Pfitzmann, and P. D. McDaniel, Eds. ACM, 2004, pp. 132–145. [Online]. Available: https://doi.org/10.1145/1030083.1030103

[48] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 552–565. [Online]. Available: https://doi.org/10.1007/3-540-45682-1_32

[49] K. G. Paterson and T. van der Merwe, "Reactive and proactive standardisation of TLS," in *Security Standardisation Research - Third International Conference, SSR 2016, Gaithersburg, MD, USA, December 5-6, 2016, Proceedings*, ser. Lecture Notes in Computer Science, L. Chen, D. A. McGrew, and C. J. Mitchell, Eds., vol. 10074. Springer, 2016, pp. 160–186. [Online]. Available: https://doi.org/10.1007/978-3-319-49100-4_7

[50] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, ser. Lecture

Notes in Computer Science, S. Vaudenay, Ed., vol. 4004. Springer, 2006, pp. 409–426. [Online]. Available: https://doi.org/10.1007/11761679_25

[51] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," *IACR Cryptol. ePrint Arch.*, vol. 2004, p. 332, 2004. [Online]. Available: http://eprint.iacr.org/2004/332

[52] D. Basin, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Formal Methods for Security, version 1.0. [Online]. Available: https://www.cybok.org/

[53] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-aided cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1393, 2019. [Online]. Available: https://eprint.iacr.org/2019/1393

[54] N. Koblitz and A. Menezes, "Critical perspectives on provable security: Fifteen years of "another look" papers," *Adv. Math. Commun.*, vol. 13, no. 4, pp. 517–558, 2019. [Online]. Available: https://doi.org/10.3934/amc.2019034

[55] E. Barker, "Recommendation for Key Management: Part 1 – General," May 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

[56] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "DROWN: breaking TLS using SSLv2," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 689–706. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram

[57] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. Béguelin, and P. Zimmermann, "Imperfect forward secrecy: how Diffie-Hellman fails in practice," *Commun. ACM*, vol. 62, no. 1, pp. 106–114, 2019. [Online]. Available: https://doi.org/10.1145/3292035

[58] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and J. K. Zinzindohoue, "A messy state of the union: taming the composite state machines of TLS," *Commun. ACM*, vol. 60, no. 2, pp. 99–107, 2017. [Online]. Available: https://doi.org/10.1145/3023357

[59] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/

[60] D. Ott and C. Peikert, "Identifying research challenges in post quantum cryptography migration and cryptographic agility," Computing Community Consortium Technical Report, 2019. [Online]. Available: https://cra.org/ccc/wp-content/uploads/sites/2/2018/11/CCC-Identifying-Research-Challenges-in-PQC-Workshop-Report.pdf

[61] T. R. Johnson, "American cryptology during the cold war, 1945-1989. book III: Retrenchment and reform, 1972-1980," 1998. [Online]. Available: https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/cryptologic-histories/cold_war_iii.pdf

[62] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators ," January 2012. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-90a.pdf

[63] VCAT, "NIST cryptographic standards and guidelines development process — report and recommendations of the Visiting Committee on Advanced

Technology of the National Institute of Standards and Technology," July 2014. [Online]. Available: https://www.nist.gov/system/files/documents/2017/05/09/VCAT-Report-on-NIST-Cryptographic-Standards-and-Guidelines-Process.pdf

[64] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994, pp. 124–134. [Online]. Available: https://doi.org/10.1109/SFCS.1994.365700

[65] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, G. L. Miller, Ed. ACM, 1996, pp. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[66] M. Bellare, T. Kohno, and C. Namprempre, "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 2, pp. 206–241, 2004. [Online]. Available: https://doi.org/10.1145/996943.996945

[67] M. Green and M. Smith, "Developers are not the enemy!: The need for usable security APIs," *IEEE Secur. Priv.*, vol. 14, no. 5, pp. 40–46, 2016. [Online]. Available: https://doi.org/10.1109/MSP.2016.111

[68] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 104–113. [Online]. Available: https://doi.org/10.1007/3-540-68697-5_9

[69] S. Ruhault, "SoK: Security models for pseudo-random number generators," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 506–544, 2017. [Online]. Available: https://doi.org/10.13154/tosc.v2017.i1.506-544

[70] P. L. Gorski, Y. Acar, L. L. Iacono, and S. Fahl, "Listen to developers! A participatory design study on security warnings for cryptographic APIs," in *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, R. Bernhaupt, F. F. Mueller, D. Verweij, J. Andres, J. McGrenere, A. Cockburn, I. Avellino, A. Goguey, P. Bjøn, S. Zhao, B. P. Samson, and R. Kocielnik, Eds. ACM, 2020, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3313831.3376142

[71] M. A. Sasse and A. Rashid, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Human Factors, version 1.0.1. [Online]. Available: https://www.cybok.org/

[72] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003. [Online]. Available: https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical

[73] J. Jancar, V. Sedlacek, P. Svenda, and M. Sýs, "Minerva: The curse of ECDSA nonces systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 281–308, 2020. [Online]. Available: https://doi.org/10.13154/tches.v2020.i4.281-308

[74] S. Weiser, D. Schrammel, L. Bodner, and R. Spreitzer, "Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 1767–1784. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/weiser

[75] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password interception in a SSL/TLS channel," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International*

*Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 583–599. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_34

[76] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[77] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel," *IACR Cryptol. ePrint Arch.*, vol. 2002, p. 169, 2002. [Online]. Available: http://eprint.iacr.org/2002/169

[78] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed., vol. 3860. Springer, 2006, pp. 1–20. [Online]. Available: https://doi.org/10.1007/11605805_1

[79] F. Tramèr, D. Boneh, and K. Paterson, "Remote side-channel attacks on anonymous transactions," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 2739–2756. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/tramer

[80] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. A. Smolin, "Experimental quantum cryptography," in *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*, ser. Lecture Notes in Computer Science, I. Damgård, Ed., vol. 473. Springer, 1990, pp. 253–265. [Online]. Available: https://doi.org/10.1007/3-540-46877-3_23

[81] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Springer, 1997, pp. 37–51. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_4

[82] Y. Kim, R. Daly, J. S. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 2014, pp. 361–372. [Online]. Available: https://doi.org/10.1109/ISCA.2014.6853210

[83] F. Piessens, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Software Security, version 1.0.1. [Online]. Available: https://www.cybok.org/

[84] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, ser. Lecture Notes in Computer Science, K. Kurosawa, Ed., vol. 4833. Springer, 2007, pp. 29–50. [Online]. Available: https://doi.org/10.1007/978-3-540-76900-2_3

[85] T. Pornin, "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," RFC 6979 (Informational), RFC Editor, Fremont, CA, USA, Aug. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6979.txt

[86] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your ps and

qs: Detection of widespread weak keys in network devices," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, T. Kohno, Ed. USENIX Association, 2012, pp. 205–220. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger

[87] K. Mowery, M. Y. C. Wei, D. Kohlbrenner, H. Shacham, and S. Swanson, "Welcome to the entropics: Boot-time entropy in embedded devices," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 589–603. [Online]. Available: https://doi.org/10.1109/SP.2013.46

[88] T. Ristenpart and S. Yilek, "When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010. [Online]. Available: https://www.ndss-symposium.org/ndss2010/when-good-randomness-goes-bad-virtual-machine-reset-vulnerabilities-and-hedging-deployed

[89] E. Barker and W. C. Barker, "Recommendation for Key Management: Part 2 – Best Practices for Key Management Organizations," May 2019. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt2r1.pdf

[90] E. Barker and Q. Dang, "Recommendation for Key Management: Part 3 – Application-Specific Key Management Guidance," January 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

[91] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446 (Proposed Standard), RFC Editor, Fremont, CA, USA, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446.txt

[92] T. Jager, K. G. Paterson, and J. Somorovsky, "One bad apple: Backwards compatibility attacks on state-of-the-art cryptography," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013. [Online]. Available: https://www.ndss-symposium.org/ndss2013/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography

[93] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869 (Informational), Internet Engineering Task Force, May 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5869.txt

[94] C. Percival and S. Josefsson, "The scrypt Password-Based Key Derivation Function," RFC 7914 (Informational), RFC Editor, Fremont, CA, USA, Aug. 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7914.txt

[95] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10212, 2017, pp. 33–62. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_2

[96] S. Katzenbeisser, Ü. Koçabas, V. Rozic, A. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon," in *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, ser. Lecture Notes in Computer Science, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, 2012, pp. 283–301. [Online]. Available: https://doi.org/10.1007/978-3-642-33027-8_17

[97] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Public keys," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology*

*Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 626–642. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_37

[98] M. Nemec, M. Sýs, P. Svenda, D. Klinec, and V. Matyas, "The return of Coppersmith's attack: Practical factorization of widely used RSA moduli," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1631–1648. [Online]. Available: https://doi.org/10.1145/3133956.3133969

[99] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson, "The matter of Heartbleed," in *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, C. Williamson, A. Akella, and N. Taft, Eds. ACM, 2014, pp. 475–488. [Online]. Available: https://doi.org/10.1145/2663716.2663755

[100] S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot, "White-box cryptography and an AES implementation," in *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, ser. Lecture Notes in Computer Science, K. Nyberg and H. M. Heys, Eds., vol. 2595. Springer, 2002, pp. 250–270. [Online]. Available: https://doi.org/10.1007/3-540-36492-7_17

[101] K. Cohn-Gordon, C. J. F. Cremers, and L. Garratt, "On post-compromise security," in *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*. IEEE Computer Society, 2016, pp. 164–178. [Online]. Available: https://doi.org/10.1109/CSF.2016.19

[102] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), RFC Editor, Fremont, CA, USA, May 2008, updated by RFCs 6818, 8398, 8399. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5280.txt

[103] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962 (Experimental), RFC Editor, Fremont, CA, USA, Jun. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6962.txt

[104] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC 6960 (Proposed Standard), RFC Editor, Fremont, CA, USA, Jun. 2013, updated by RFC 8954. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6960.txt

[105] J. Larisch, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A scalable system for pushing all TLS revocations to all browsers," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 539–556. [Online]. Available: https://doi.org/10.1109/SP.2017.17

[106] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 17–36. [Online]. Available: https://doi.org/10.1007/11535218_2

[107] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10401. Springer,

2017, pp. 570–596. [Online]. Available: https://doi.org/10.1007/978-3-319-63688-7_19

[108] G. Leurent and T. Peyrin, "From collisions to chosen-prefix collisions application to full SHA-1," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11478. Springer, 2019, pp. 527–555. [Online]. Available: https://doi.org/10.1007/978-3-030-17659-4_18

[109] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: A usability evaluation of PGP 5.0," in *Proceedings of the 8th USENIX Security Symposium, Washington, DC, USA, August 23-26, 1999*, G. W. Treese, Ed. USENIX Association, 1999. [Online]. Available: https://www.usenix.org/conference/8th-usenix-security-symposium/why-johnny-cant-encrypt-usability-evaluation-pgp-50

[110] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, ser. Lecture Notes in Computer Science, G. R. Blakley and D. Chaum, Eds., vol. 196. Springer, 1984, pp. 47–53. [Online]. Available: https://doi.org/10.1007/3-540-39568-7_5

[111] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, ser. Lecture Notes in Computer Science, C. Laih, Ed., vol. 2894. Springer, 2003, pp. 452–473. [Online]. Available: https://doi.org/10.1007/978-3-540-40061-5_29

[112] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 289–305. [Online]. Available: https://doi.org/10.1109/SP.2016.25

[113] ETSI, "Security architecture and procedures for 5G System (3GPP TS 33.501 version 15.4.0 Release 15)." [Online]. Available: https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts_133501v150400p.pdf

[114] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 451–466. [Online]. Available: https://doi.org/10.1109/EuroSP.2017.27

[115] C. Troncoso, M. Payer, J. Hubaux, M. Salathé, J. R. Larus, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, L. Barman, S. Chatel, K. G. Paterson, S. Capkun, D. A. Basin, J. Beutel, D. Jackson, M. Roeschlin, P. Leu, B. Preneel, N. P. Smart, A. Abidin, S. Gurses, M. Veale, C. Cremers, M. Backes, N. O. Tippenhauer, R. Binns, C. Cattuto, A. Barrat, D. Fiore, M. Barbosa, R. Oliveira, and J. Pereira, "Decentralized privacy-preserving proximity tracing," *IEEE Data Eng. Bull.*, vol. 43, no. 2, pp. 36–66, 2020. [Online]. Available: http://sites.computer.org/debull/A20june/p36.pdf

[116] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 483–502. [Online]. Available: https://doi.org/10.1109/SP.2017.26

[117] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A comprehensive symbolic analysis of TLS 1.3," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM,

2017, pp. 1773–1788. [Online]. Available: https://doi.org/10.1145/3133956.3134063

[118] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1044, 2020. [Online]. Available: https://eprint.iacr.org/2020/1044

[119] N. Drucker and S. Gueron, "Selfie: reflections on TLS 1.3 with PSK," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 347, 2019. [Online]. Available: https://eprint.iacr.org/2019/347

[120] F. Günther, B. Hale, T. Jager, and S. Lauer, "0-RTT key exchange with full forward secrecy," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10212, 2017, pp. 519–548. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_18

[121] N. Aviram, K. Gellert, and T. Jager, "Session resumption protocols and efficient forward security for TLS 1.3 0-RTT," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11477. Springer, 2019, pp. 117–150. [Online]. Available: https://doi.org/10.1007/978-3-030-17656-3_5

[122] D. Derler, K. Gellert, T. Jager, D. Slamanig, and C. Striecks, "Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange," *J. Cryptol.*, vol. 34, no. 2, p. 13, 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09374-3

[123] F. Dallmeier, J. P. Drees, K. Gellert, T. Handirk, T. Jager, J. Klauke, S. Nachtigall, T. Renzelmann, and R. Wolf, "Forward-secure 0-RTT goes live: Implementation and performance analysis in QUIC," in *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, S. Krenn, H. Shulman, and S. Vaudenay, Eds., vol. 12579. Springer, 2020, pp. 211–231. [Online]. Available: https://doi.org/10.1007/978-3-030-65411-5_11

[124] C. Garman, M. Green, G. Kaptchuk, I. Miers, and M. Rushanan, "Dancing on the lip of the volcano: Chosen ciphertext attacks on Apple imessage," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 655–672. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/garman

[125] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From keys to databases – real-world applications of secure multi-party computation," *Comput. J.*, vol. 61, no. 12, pp. 1749–1771, 2018. [Online]. Available: https://doi.org/10.1093/comjnl/bxy090

[126] A. Viand, P. Jattke, and A. Hithnawi, "SoK: Fully homomorphic encryption compilers," *CoRR*, vol. abs/2101.07078, 2021. [Online]. Available: https://arxiv.org/abs/2101.07078

[127] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 370–389. [Online]. Available: https://doi.org/10.1109/EuroSP48549.2020.00031

[128] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, A. Akella and J. Howell, Eds. USENIX Association, 2017, pp. 259–282. [Online]. Available: https://

www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs

[129] T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How not to prove your election outcome," in *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 644–660. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00048

# ACRONYMS

| | |
|---|---|
| **AD** | Associated Data. |
| **AE** | Authenticated Encryption. |
| **AEAD** | Authenticated Encryption with Associated Data. |
| **AKE** | Authenticated Key Exchange. |
| **API** | Application Programming Interface. |
| **AuC** | Authentication Centre. |
| **CA** | Certification Authority. |
| **CaaS** | Cryptography-as-a-Service. |
| **CDHP** | Computational Diffie-Hellman Problem. |
| **CFRG** | Crypto Forum Research Group. |
| **CPU** | Central Processing Unit. |
| **CRL** | Certificate Revocation List. |
| **CRT** | Chinese Remainder Theorem. |
| **CT** | Certificate Transparency. |
| **DEK** | Data Encryption Key. |
| **DEM** | Data Encapsulation Mechanism. |
| **DH** | Diffie-Hellman. |
| **DHIES** | Diffie-Hellman Integrated Encryption Scheme. |
| **DHKE** | Diffie-Hellman Key Exchange. |
| **DLP** | Discrete Logarithm Problem. |
| **DSKS** | Duplicate Signature Key Selection. |
| **ECC** | Elliptic Curve Cryptography. |
| **ECIES** | Elliptic Curve Integrated Encryption Scheme. |
| **FHE** | Fully Homomorphic Encryption. |
| **GAEN** | Google-Apple Exposure Notification. |

|  |  |
|---|---|
| **GPU** | Graphical Processing Unit. |
| **HSM** | Hardware Security Module. |
| **IBC** | Identity-Based Cryptography. |
| **IETF** | Internet Engineering Task Force. |
| **IFP** | Integer Factorisation Problem. |
| **IND-CCA** | Indistinguishability under Chosen Ciphertext Attack. |
| **IND-CPA** | Indistinguishability under Chosen Plaintext Attack. |
| **INT-CTXT** | Integrity of Ciphertexts. |
| **IoT** | Internet of Things. |
| **IRTF** | Internet Research Task Force. |
| **KDF** | Key Derivation Function. |
| **KEK** | Key Encryption Key. |
| **KEM** | Key Encapsulation Mechanism. |
| **MAC** | Message Authentication Code. |
| **MitM** | Man-in-the-Middle. |
| **MPC** | Multi-Party Computation. |
| **NIST** | National Institute for Standards and Technology. |
| **NSA** | National Security Agency. |
| **OCSP** | Online Certificate Status Protocol. |
| **PBKDF** | Password-based Key Derivation Function. |
| **PKE** | Public Key Encryption. |
| **PKI** | Public Key Infrastructure. |
| **PQC** | Post-quantum Cryptography. |
| **PRF** | Pseudo-Random Function. |
| **PRNG** | Pseudo-Random Number Generator. |
| **PRP** | Pseudo-Random Permutation. |
| **PUF** | Physically Unclonable Function. |
| **QKD** | Quantum Key Distribution. |
| **SGX** | Software Guard eXtensions. |

**SIM** Subscriber Identity Module.

**SSL** Secure Sockets Layer.

**SUF-CMA** Strong Unforgeability under Chosen Message Attack.

**TA** Trusted Authority.

**TLS** Transport Layer Security.

**TPM** Trusted Platform Module.

**TRBG** True Random Bit Generator.

**TRNG** True Random Number Generator.

**TTP** Trusted Third Party.

**VM** Virtual Machine.

# INDEX

5G network, 47

A5/1 stream cipher, 10
A5/2 stream cipher, 10
access control, 38
AE Variants, 13
AES, 10, 11, 13, 14, 20, 24, 30, 32, 50, 51
AES-GCM, 11, 13, 14, 30
AES-GCM-SIV, 13
AMD, 14, 34
Android, 39
Android Keystore, 39
anonymity, 5, 15, 20, 32, 52
Apple, 44, 49, 51
Apple iMessage, 49
application layer, 47
application programming interface, 4, 28–30, 39, 46
applied cryptography, 4, 6–8, 20–22, 35, 48, 52, 53
Argon2, 37
ARM, 32, 39
ARM TrustZone, 32, 39
arms race, 39
ASCII, 30
assurance of purpose, 36
attack surface, 5
attack vector, 5
attacker motivation, 20
attestation, 7, 20
authenticated encryption, 4, 10, 11, 13
authenticated encryption with associated data, 13, 14, 27, 29, 30, 36, 40, 41, 48
authenticated key exchange, 18, 37
authentication, 4, 7, 10, 11, 13, 18–20, 26, 37, 40, 41, 44, 48–50
authentication centre, 40
authenticity, 27, 40, 42, 45, 49

backdoor, 6
backward security, 41
bandwidth, 39, 44, 53
banking services, 37
base station, 47
beacon, 51
birthday paradox, 9
Bitcoin, 21
Bleichenbacher attack, 16
blind signature scheme, 20
block cipher, 9–11, 14, 20, 29, 31, 37, 51
blockchain, 52
Bluetooth, 51
Bluetooth Low Energy, 51
boot time entropy hole, 34
BoringSSL, 28
BouncyCastle, 28
Browser Forum, 43
buffer, 32, 39
bulletin board, 42

C, 28
cache, 10, 31, 38, 50
cache attacks, 32
Caesar cipher, 52
CAESAR competition, 14
centralisation, 40, 51
certificate, 36, 42–44
certificate authority, 42–44
certificate chain, 42–44
certificate revocation list, 44
Certificate Transparency, 43, 44
ChaCha20, 14, 34
ChaCha20-Poly1305, 14
chain of keys, 40, 51
chaos-based cryptography, 45
child abuse, 6
chinese remainder theorem, 16, 33
chosen message attack, 11, 14, 19, 27
chosen plaintext attack, 12–16, 27, 34
chosen-prefix collision attack, 44
cipher block chaining, 14, 31, 50
ciphertext, 9–16, 18, 26, 27, 30, 31, 41, 45
classical computer, 25
cloud computing, 5, 7, 31, 38, 39, 47
CloudFlare, 26
code repository, 42
collision resistance, 9, 44
commitment scheme, 32
communication channel, 7
communication protocol, 4, 5, 14, 24, 33, 47, 48
compiler, 34

complexity theory, 22
composition theorem, 16
computational complexity, 13, 22, 23
computational Diffie-Hellman problem, 17, 18
computationally hard problem, 4, 9, 16, 17, 22
concrete reduction, 22
confidentiality, 4, 5, 8, 11–13, 27, 48
consensus, 24
constant-time cryptography, 33
contact tracing, 51, 52
correctness, 12, 14, 21, 26, 42
counter mode, 14, 51
COVID-19, 51
CPU, 10, 14, 21, 31, 32, 34, 38
credentials, 46, 47
CRLite, 44
cryptanalysis, 5, 21, 23, 44, 46
Crypto Forum Research Group, 25
cryptocurrency, 5, 6, 15, 20, 21, 32, 52
cryptocurrency mining, 21
cryptograhpic protocols, 18, 22, 24, 27, 53
cryptographic agility, 21, 23, 24, 44
cryptographic assumption, 5, 11, 14, 17, 19, 21, 22, 27, 36, 51, 53
cryptographic libraries, 4, 28, 29, 46, 47, 49, 52
cryptographic primitives, 4, 8, 28, 46, 48, 50
cryptographic scheme, 8, 9, 16, 21, 22, 26, 28, 35, 45, 46
cryptographically opinionated, 24
cryptography, 4–8, 17, 20–22, 24–26, 28, 29, 32–37, 39, 40, 42, 45–48, 51–53
cryptography-as-a-service, 47
culture, 45

data at rest, 7, 52
data encapsulation mechanism, 16, 18
data encryption key, 39
data in transit, 7, 52
data mining, 53
data origin authentication, 11, 19
data security, 8
data source, 44
data storage, 7
data stream, 10, 48
data under computation, 7, 52
database, 7, 37, 53
de-obfuscation tools, 39
Debian, 38
decryption failure, 12
DES, 9, 25
deterministic algorithm, 5, 11, 15, 19, 53
developers, 4, 10, 24, 28, 29, 39, 46, 47, 52
development, 6, 21, 23, 25, 28, 29, 44, 47, 52
Diffie-Hellman integrated encryption scheme, 18
Diffie-Hellman key exchange, 4, 17, 18, 36, 41, 48, 50
DigiNotar, 43
digital signature, 4, 15, 18, 19, 26, 41, 42, 49
direct anonymous attestation, 20
discrete logarithm problem, 17, 23, 25, 41
diversification, 37
diversity, 20
documentation, 29
domain-specific, 33
double ratcheting, 50
DP-3T, 51, 53
drug dealing, 6
dummy packets, 33
duplicate signature key selection attack, 19
dynamic analysis, 33

e-voting, 20, 53
E&M scheme, 5, 14
eavesdropping, 17, 26, 47
economics, 5
EdDSA, 19
electro-magnetic, 33
electronic circuit, 38
electronic codebook, 29
ElGamal encryption, 18, 23, 45
elliptic curve cryptography, 17–19, 25, 31, 34, 41
Elliptic Curve digital signature algorithm, 19, 31, 34
embedded systems, 34
enclaves, 39, 52
encoding, 16, 17, 19, 25, 44
Encrypt-and-MAC, 14
Encrypt-then-MAC, 14
encrypted database, 53
encryption, 4, 5, 8–19, 27, 29–31, 33, 34, 36, 37, 39–41, 45–48, 50, 53
end-to-end security, 5, 26, 48
entropy, 34–37
entropy extraction, 36
entropy pool, 34, 35
entropy sources, 34
ephemeral key, 41
epidemiological, 51

error side channel, 31
ETSI, 25, 26
execution time, 34
exhaustive key search, 9, 20, 25, 36, 37
exploit, 52
export cipher, 24
export restriction, 6
extended Euclidean algorithm, 16

factoring problem, 5, 16
fault attacks, 33
FIDO alliance, 7
financial sector, 39, 40
fingerprinting, 50
finite field, 14, 18, 19, 23, 25, 41, 50
FIPS 140-1 / FIPS, 39
first crypto war, 6
first-order approximation, 7
Flush+Reload attack, 32
formal definition, 5, 21, 27
forward security, 40, 41, 48–50
fully homomorphic encryption, 5
fuzz testing, 33

game hopping, 22
generic composition, 5, 14, 27
glitch attack, 33
GMAC, 11
Google, 26, 28, 43, 47, 48, 51, 53
Google Chrome, 47
Google-Apple Exposure Notification system, 51
government, 6, 10, 24, 43, 52
GPU, 21, 37
group signature scheme, 20
Grover's algorithm, 25

hardware acceleration, 32
hardware design, 33
hardware security, 33, 39
hardware security module, 39, 40, 47
hash chain, 40
hash function, 8, 9, 11, 14, 17, 19–23, 30, 37, 40, 50
hashed passwords, 30, 37
heartbeat protocol, 39
Heartbleed, 28, 38
HMAC, 11, 37, 50
homomorphic encryption, 5
HSM-as-a-service, 47
HTTP, 48
human rights, 6
human-readable, 50
hybrid encryption, 14

identity-based cryptography, 45
IEC, 24
IETF, 24, 26, 48
impersonation, 42, 49
IND-CCA security, 13, 15–17, 32
IND-CPA security, 12–16, 27, 34
indistinguishable encryption, 12
information flow, 33
information leakage, 4, 5, 7, 12, 15, 19, 30–33, 36, 53
information security, 35, 43, 46
information-theoretic security, 36
initialisation vector, 10
input space, 9
instant messaging, 6, 48, 49, 52
instruction set architecture, 33
integrity, 4, 5, 11–13, 15, 18, 19, 27, 32, 48
integrity of ciphertexts, 12
integrity of plaintexts, 13
Intel, 14, 32, 34, 39
Intel SGX, 32, 39, 52
interactive protocol, 8, 27
international standard, 24
internet, 24, 25, 28, 32, 52
interoperability, 24, 25
interview, 29
iOS, 39
iOS Secure Enclave, 39
IoT, 32, 42, 43, 52
IPsec, 24, 33
IRTF, 25, 37
ISO, 24
isolation, 31

Java, 28

key derivation, 18, 35–37, 50
key derivation function, 18, 35–37, 50
key distribution, 26
key encapsulation mechanism, 14, 16–18, 25, 26, 41
key encryption key, 39
key generation, 8, 11, 14, 18, 36, 38
key hierarchy, 37
key length, 9, 38

key life-cycle, 35, 36
key management, 4, 6, 7, 27, 35, 42, 46, 47
key pair, 8, 14, 15, 18, 19, 34, 38
key-stream, 10
keyboard, 34
keygen algorithm, 8, 11, 14, 15, 18, 22, 34, 38
keyless encoding, 16

latency, 49
laws of physics, 26, 33
left-or-right encryption oracle, 12, 13, 15
legacy system, 21, 24, 37, 53
legal authority, 41
legal framework, 52
legislation, 52
length side channel, 30, 33
LibreSSL, 28
license, 28
linear feedback shift register, 10
location data, 51
logging, 43
logical network, 40

MAC-then-Encrypt, 14
machine language, 22
machine learning, 5, 53
magnetic disks, 40
man-in-the-middle attack, 18, 41, 50
manufacturing, 39
master secret, 37
mathematics, 4, 8, 14, 19, 22, 45
MD5, 23, 40
memory access, 32, 34
memory sharing, 31
mental models, 29
message authentication code, 5, 11, 18, 19, 32, 37, 50
message exchange, 8, 49
metadata, 30, 51
methodology, 29, 39
micro-architecture, 32
military, 6
millionaire's problem, 5
mitigating risk, 5
mobile app, 42
mode of operation, 10, 14
Monero, 20
money laundering, 6
Moore's law, 5
Mozilla Firefox, 44, 47, 53
multi-party computation, 5, 52, 53
multi-user system, 13

NIST, 9, 10, 20, 23–26, 34, 35, 39
non-repudiation, 18, 19
nonce, 11–14, 27, 29, 34
nonce respecting adversary, 13
NSA, 23, 25
number field sieve, 5, 23
number theory, 4

obfuscation, 39
OCSP stapling, 44
open source, 28, 49
OpenBSD, 28
OpenSSL, 28, 29, 39
Operating System, 34, 35, 38, 39
oracle, 9, 12, 13, 15, 17, 31
out-of-band distribution, 42
outsourced data, 7, 52
outsourced database, 7

paper tape, 40
parallelism, 5, 31
passive attack, 39, 41
password, 30, 37, 38, 45
password authenticated key exchange, 37
payment card industry, 32
perfect isolation, 31
PGP protocol, 45
physical proximity, 51
physically unclonable function, 38
PKCS#1 v1.5, 16, 19, 25, 40
plaintext, 5, 9–16, 27, 30–33, 46, 52
policies, 44
politics, 6
polynomial time, 22
post-quantum, 20, 23–26
pre-image resistance, 9
primality test, 18, 29
principle of key separation, 36, 37
Prio, 53
privacy, 5–7, 20, 44, 48, 51–53
privacy-preserving machine learning, 5
privacy-preserving outsouring, 7
privacy-preserving systems, 51, 53
private key, 8, 14–16, 18, 32, 33, 39, 41–43, 45
probabilistic signature scheme, 19, 34
probability, 9, 12, 13, 15
process id, 34

processes, 6, 28, 31, 35, 36, 38, 44
production environment, 33
programming language, 4, 22, 28
pseudo-code, 22, 28, 30
pseudo-random, 9, 34, 36, 38, 51
pseudo-random function, 36
pseudo-random number generator, 34, 35
pseudo-random permutation, 9, 11, 14
public key, 4, 7, 8, 14–17, 20, 23, 25, 26, 34, 36, 40–45, 49, 52
public key cryptography, 4, 7, 17, 25, 40, 42, 45, 52
public key encryption scheme, 8, 14–19, 22, 25, 32, 34, 35, 41, 49
public key infrastructure, 42

quantum computers, 21, 23, 25
quantum key distribution, 26, 33
quantum-safe, 25

random oracle model, 9
random seed, 34, 38
randomised algorithm, 8, 11, 14, 15, 34, 38
ransom payment, 6
RC4 stream cipher, 10
redundancy, 10
regulation, 6
remote attack, 31
replay attack, 49
request for comments, 24
resource partitioning, 38
responsibility, 13
reverse engineering, 49
revoke keys, 36
ring signature, 20
robustness, 15
root certificate, 42
root certificate authority, 43
Rowhammer, 33
RSA, 16–19, 23, 25, 33, 34, 40, 41, 45, 48, 50
RSA inversion problem, 17
RSA transform, 16

salting passwords, 37
scalability, 7
Schnorr signature, 19
scrypt, 37
secret key, 14, 18
secure channel, 6, 27, 35, 48
secure socket layer, 10, 14, 24, 31–33
secure storage, 7, 36, 39
security bug, 29
security failure, 6, 29, 47
security goal, 4, 29, 30
security policies, 44
security practitioner, 4, 6
security proof, 17, 19, 21, 22, 30, 49, 53
security requirements, 10, 29, 38, 42
sensitive information, 5, 30–33, 52
sequence number, 6
sequences of games, 22
session key, 35, 40–42, 48
SHA-1, 9, 40, 44
SHA-256, 9, 17, 50
SHA-3, 9, 24
Shor's algorithm, 25
side-channel vulnerability, 4
Signal, 6, 48–50, 52
signal protocol, 49
signature scheme, 18–20, 34, 41, 44
signcryption, 15, 49
SIM, 40
SIM cards, 40
small sub-group attack, 18
smart-cards, 33, 35, 38, 39
Snowden revelations, 6
social processes, 6
social science, 29
software as a service, 47
Software Guard Extensions, 32, 39, 52
software security, 33
software update, 42, 43
SPHINCS+, 20
spoofing, 41, 44
square-and-multiply algorithm, 16, 33
SSH, 6, 14, 24
Stack Exchange, 47
stakeholder, 25
standardisation, 16, 23–25, 37
static analysis, 33
stream cipher, 10
strong typing, 33
strong unforgeability, 11, 14, 19, 27
strong unforgeability under chosen message attack, 11, 14, 19, 27
surveillance, 43
symbolic model, 28
symmetric encryption, 5, 11, 14, 29, 50, 53
synchronisation, 41, 50

system designer, 5, 40

tampering, 27
telecommunications, 10, 40
Telegram, 6, 49, 50
telemetry, 53
terrorism, 6
textbook, 8, 16, 19
the crypto wars, 6
third-party software, 38
timing attacks, 31
trading platform, 30
traffic analysis, 5, 30
transparency, 25, 43, 44
Transport Layer Security, 4, 10, 14, 21, 24, 26, 31–33, 35, 39–41, 43, 44, 47–49, 52, 53
tree of keys, 37
trial decryption, 15, 32
triumvirate, 7, 52
true random bit generator, 34
trust, 18, 20, 32, 35, 42, 43, 45, 47, 49, 50, 52
trusted authority, 45
trusted bulletin board, 42
Trusted Computing Group, 20, 32
trusted platform module, 20, 32, 39
trusted third party, 42, 45
TurkTrust, 43

ubiquitous, 47
unconditional security, 20, 26, 33
uniformly random, 9, 17, 34, 38
unique key per transaction, 37
universal hash function, 11, 14
URL, 43
usability, 4, 29, 45–47, 53
usable security, 29
use case, 13, 14, 28, 30, 49

virtual machine, 34
virtualisation, 31
vulnerabilities, 4, 10, 19, 24, 28, 29, 33, 36, 38, 39, 49, 52, 53

web browser, 38, 43, 44, 47, 53
web client, 49
web of trust, 45
web public key infrastructure, 43, 44, 48, 49
web server, 44
website, 38, 43, 47
WEP, 10
WhatsApp, 49, 50
Wheeler's aphorism, 7
white-box cryptography, 39
WireGuard, 24
wireless communication, 10
wireless router, 40
WPA, 10

X.509 certificate, 42, 44
XOR, 10

Zcash, 32
zero round trip, 48, 49
zero-knowledge, 5, 32, 52
zero-knowledge proof, 5, 32, 52