Traditional and Emerging Data Protection Methods

Marina Blanton

Abstract This chapter treats data protection mechanisms not covered in other parts of the book and examines whether they can be used to protect biometric templates in verification and identification applications. The considered mechanisms include cryptographic, non-cryptographic, and hardware-based solutions, which are divided into traditional and emerging mechanism categories. The chapter demonstrates that many of these protection mechanisms are not suitable as solutions for biometric template protection. For example, cryptographic constructions such as encryption schemes and hash functions possess strong security properties; however, the strong obfuscation they achieve by erasing all patterns from the input is the reason they are not suitable for working with noisy biometric data. We also discuss potential solutions that rely on distributed databases, smart cards, and trusted execution environments (TEE) and only consider TEE as conceptually suitable for achieving the necessary security guarantees for protecting biometric templates in verification and identification applications.

1 Introduction

The second part of this book describes techniques and solutions that can be used to protect biometric templates in various applications. The goal of this chapter is to outline several data protection mechanisms not covered in other chapters. In particular, many conventional tools used for data protection, e.g., hash functions or encryption schemes, can be thought of as potential solutions for protecting biometric templates. However, while these tools can be used for achieving strong security properties in other contexts, e.g., for protecting data at rest, they fall short of achiev-

Marina Blanton
University at Buffalo, Buffalo, USA
e-mail: mblanton@buffalo.edu

ing the necessary security guarantees when used for biometric-based verification or identification.

In this chapter, we discuss traditional data protection mechanisms, including the use of hashing, conventional encryption, distributed databases, and smart cards. Section 2 describes these techniques, as well as why they are, on their own, unable to achieve the desired security properties. We also discuss the use of an emerging technology, Trusted Execution Environment (TEE), as a potential solution to protecting biometric templates. This material is covered in Section 3. We afterwards conclude and provide questions for further study.

2 Traditional Data Protection Mechanisms

This section discusses traditional data protection mechanisms. We begin with definitions of the relevant methods and then proceed to show why they fall short of achieving the desired security properties for biometric-based verification or identification. We start with cryptographic solutions that use hash-based techniques and conventional encryption, and this is followed by a discussion on distributed databases and smart cards.

2.1 Hashing

A cryptographic hash function is an algorithm that, given a string of arbitrary size, produces a digest that must satisfy certain security properties. More formally, let $h : \{0,1\}^* \rightarrow \{0,1\}^k$ be a hash function that takes an input of arbitrary size and produces a k-bit output. A cryptographic hash function must meet the following security properties:

1. *Preimage resistance:* Let y be the output of h on some input. Given h and y, it is infeasible to find any x such that $h(x) = y$.
2. *Second preimage resistance:* Given h and x, it is infeasible to find x' such that $x' \neq x$ and $h(x') = h(x)$.
3. *Collision resistance:* Given h, it is infeasible to find x and x' such that $x' \neq x$ and $h(x') = h(x)$.

When we say that it is infeasible to find x or x' to violate one of the security properties above, we refer to this term's formulation in a rigorous cryptographic sense. That is, given a security parameter λ, we consider an adversary who can perform arbitrary computation and employ any strategy of their choice in breaking the security. The important assumption is that the adversary is computationally limited and is constrained to run in polynomial time in the security parameter λ.

Definition 1 A function is said to be *negligible* in security parameter λ, denoted by $\text{negl}(\lambda)$, if it diminishes faster than the inverse of any polynomial function in λ.

More precisely, a function $f(\cdot)$ is negligible if for every polynomial $p(\cdot)$ there exists a constant c such that for all $\lambda > c$ it holds that $f(\lambda) < \frac{1}{p(\lambda)}$.

A simple example of a negligible function is $\frac{1}{2^\lambda}$, while more generally any function of the form $\frac{1}{g(\lambda)}$ for a super-polynomial $g(\cdot)$ is negligible. This means that, having control over the security parameter λ, it is always possible to reduce $\mathrm{negl}(\lambda)$ to any sufficiently small value. Going back to the formulation of the security properties of a hash function, "infeasible" means that, given a security parameter λ, any probabilistic polynomial-time adversary has at most negligible probability in violating the property.

In practice, the choice of security parameter λ is driven by our computational abilities and the best known algorithms for breaking the security property in question. For example, performing work on the order of 2^{128} in a reasonable amount of time is considered today to be out of reach. In the context of hash functions, generic attacks on preimage resistance and second preimage resistance require work on the order of 2^k hash function evaluations when the hash function output is k bits long. However, achieving at least 50% probability of success for violating the collision resistance property requires time on the order of $2^{k/2}$ hash function evaluations. This dictates the choice of the hash output length k, which plays the role of the security parameter. That is, achieving 128-bit security while satisfying all three security properties dictates the use of $k = 256$ or longer for constructions without algorithm-specific weaknesses. Hash functions without algorithm-specific weaknesses are those for which there are no faster ways to violate one or more of the security properties than generic attacks, while having an algorithm-specific weakness removes a candidate hash function from consideration as having a design flaw.

Common hash function algorithms used today are SHA-256 and SHA-3 [5, 6]. SHA stands for Secure Hash Algorithms, which are a family of cryptographic hash functions published by the US National Institute of Standards and Technology (NIST). SHA-0 and its revised version SHA-1 were designed by the US government, produce 160-bit output, and are no longer in use due to both short output size and discovered weaknesses. SHA-2 is based on a similar design and supports different block sizes as specified in the names of the hash functions; e.g., SHA-256 produces 256-bit hashes, while SHA-512 has a 512-bit output. SHA-3 was adopted most recently after an open public competition process. It has a significantly different design structure from those of the rest of the SHA hash functions and also supports different hash sizes.

Cryptographic hash functions find their uses in a large number of applications. One representative and relevant application is for protecting the storage of passwords. In particular, to prevent recovery of passwords of all users stored in a system in the event of a computer compromise or privilege abuse, passwords are not stored in the clear and are instead hashed. In its simplest form, a user record could look like this:

$$\mathsf{userid},\ h(\mathsf{pwd})$$

This prevents immediate password disclosure and instead requires one to search through the password space to determine a matching password. That is, because a hash function's preimage resistance property guarantees that the hash function is one-way, recovering pwd can only be achieved by applying the hash function, h, to different password guesses. For example, to enumerate all 8-character passwords consisting only of English alphabet lower-case letters, one needs to try 26^8 hash function evaluations. A more intelligent attack that tries dictionary words with common substitutions would need to go through all passwords in the dictionary file to determine if pwd is one of them. Verification of a legitimate password, on the other hand, is simple and proceeds by hashing a user-supplied password and comparing it to the stored hash.

Now suppose that we would like to use a similar mechanism for protecting biometric templates. A fundamental challenge arises when we consider that biometric samples (and therefore templates) are noisy and thus do not result in an exact match. This is problematic because cryptographic tools, including hash functions, are designed to amplify input differences in order to produce outputs which are difficult to distinguish from random strings. This means that related biometric templates will produce hashed values with no similarities. For example, consider this 5-byte sequence (listed in binary notation) and another, related, sequence with two bits flipped (marked in bold):

10001**1**01001010010101011**1**1010101110010011
10001**0**01001010010101010**0**1010101110010011

The corresponding SHA-256 hashes (in hexadecimal) are:

978959aca040da961eba72299cd04d39bf7dadf1eeea6e9aeacfa4b8371cf974
8aa4b2c0a9a6ded735627c5da8bc54d4b5768bccccd3a360497e2d4a3c76b8fd

Therefore, since even small differences between a reference and probe biometric template would result in completely different hashes, cryptographic hashing does not work as a protection mechanism for biometric data.

Now consider that we would like to apply a hash function in a different way. Because a biometric template (after feature extraction) is represented by a number of elements, we could apply the hash function to each element of the template separately instead of to the template as a whole. That is, given a biometric template $B = (b_1, b_2, \ldots, b_n)$, we would have the server store

$$h(b_1), h(b_2), \ldots, h(b_n)$$

Then two biometric representations would be considered a match if the number of the elements equal to each other is above a predefined threshold.

With this solution, if the individual elements are sufficiently fine-grained, we are looking for exact equality, and the use of a hash function would not interfere with matching. However, this approach does not lead to adequate protection of the stored templates. That is, because each element comes from a small space, it is possible for someone with access to the database to hash all possible values for individual

elements b_i of the biometric template and consequently recover the templates stored at the server.

2.2 Conventional Encryption

General-purpose encryption provides strong data confidentiality protection against anyone without access to the decryption key. An encryption scheme is defined by three algorithms: key generation KeyGen that takes a security parameter and produces a fresh key, encryption Enc that takes a message and a key and produces a ciphertext, and decryption Dec that takes a ciphertext and a key and produces a decrypted message or outputs failure.

Encryption can be *symmetric*, which may also be called secret-key, or asymmetric, which is widely known as *public-key*. Symmetric encryption uses the same key for encryption and decryption, and thus the key must remain private. That is, given a key k generated according to the corresponding key generation algorithm, we encrypt a message m to obtain a ciphertext c as $c = \mathsf{Enc}(k,m)$, and we decrypt a ciphertext c to obtain the corresponding plaintext m as $m = \mathsf{Dec}(k,c)$.

With public-key encryption, on the other hand, key generation produces a key pair $(\mathsf{pk},\mathsf{sk})$, where pk is a public key, which is widely disseminated, and sk is the corresponding private (secret) key. Anyone with access to the public key can encrypt a plaintext message m as $c = \mathsf{Enc}(pk,m)$, while only the key owner with access to the private key sk will be able to decrypt the ciphertext c as $m = \mathsf{Dec}(sk,c)$.

Symmetric and public-key encryption have fundamentally different designs, which reflects on their speed. Symmetric encryption is normally realized using a block cipher, while public-key encryption uses number-theoretic constructions. Symmetric encryption algorithms are significantly faster than public-key encryption, while public-key cryptography is more powerful and allows for additional functionalities which cannot be realized by means of symmetric key primitives alone.

Secure encryption of any kind must meet stringent security requirements. There are different formulations of security based on the capabilities an adversary is assumed to have. However, for a general-purpose encryption algorithm to be of practical relevance, it must be secure at least against what is known as ciphertext *indistinguishability* under an *adaptive chosen plaintext attack* (CPA). The intuition is that a realistic adversary, who is allowed to receive multiple encrypted messages and influence what is being encrypted, should be unable to obtain any information about the content that a ciphertext encrypts besides the obvious observable information such as the ciphertext size. More formally, an adversary is allowed to request encryptions of messages of their choice and at some point produces two messages of the same size. One of those messages, the challenge, is encrypted and the ciphertext is given to the adversary. The adversary's goal is to determine which message from those two was encrypted.

Similar to other cryptographic functionalities, security guarantees are tied to a security parameter specified at key generation time. As before, the adversary is bound to run in polynomial time in the security parameter, and now the adversary's success beyond a random guess in determining which message was chosen as the challenge is bound by a probability negligible in the security parameter. Once again, the security parameter is guided by the best known algorithms for breaking the security of a construction. Public-key encryption algorithms require larger key and message sizes than symmetric key encryption, and this is what contributes to the performance gap.

The formulation of secure encryption above requires that no information about encrypted messages can be deduced by analyzing ciphertexts. An important implication of the above is that an adversary is unable to tell whether two different ciphertexts encrypt the same message. This means that secure encryption is necessarily randomized: a single message encrypts to a large number of possible ciphertexts using randomness drawn at the time of encryption, and a ciphertext is longer than the original plaintext.

Now, let's return to the task of securely comparing two biometric templates. For concreteness, let us assume symmetric encryption with ciphertexts of the form $c = \mathsf{Enc}(k,m)$. This can be realized by using a strong block cipher such as AES [4] in a secure encryption mode. Then encrypting a biometric template and storing the ciphertext on a server will not allow the server to discover any information about the biometric if it does not possess the key.

Now, a challenge arises when we would like to compare two biometric templates in a protected form. Because encrypting the same message multiple times produces different ciphertexts, two ciphertexts cannot be meaningfully compared to determine if they encrypt the same message. Otherwise, the encryption discloses information about whether two ciphertexts have the same plaintext content, trivially violating the necessary security guarantees.

Suppose that we degrade security to use a deterministic encryption algorithm (which on the same key-message pair always produces the same output) such as a plain invocation of a block cipher, which does not qualify as secure encryption. Even in that case, comparing two biometric templates in an encrypted form is not feasible. This, once again, stems from the fact that biometric templates are approximate and differ with each capture of the biometric sample. Cryptographic tools, on the other hand, and encryption in particular, are designed to amplify any differences in the input. Specifically in the context of block ciphers, a 1-bit difference in a plaintext block impacts all of the bits of the corresponding ciphertext block, obfuscating all information about input similarities. This is an intentional design choice of block cipher algorithms to erase any patterns from a ciphertext.

Now consider the possibility that the server is granted access to the key at the time of biometric matching: it decrypts a stored encrypted template, performs the comparison, and securely erases the decrypted value. Unfortunately, this variant does not meet the expected security guarantees. Namely, the server gains cleartext (i.e., unprotected) access to sensitive biometric data, which is no longer protected. Then any abuse of the system in the form of insider access or system compromise can lead to large-scale disclosure of sensitive biometric information.

2.3 Distributed Databases

A distributed database corresponds to multiple interconnected databases distributed across different physical locations or computers. The design goals are to improve performance (e.g., throughput and latency of content delivery to users), availability, security (e.g., fault tolerance), and scalability. In the context of this chapter, we are interested in investigating whether the use of distributed databases could also aid in protecting the privacy of biometric templates.

A possible solution could be to distribute biometric data in such a way that the entire database of biometric templates is not available on a single computer or at a single location. For example, we could partition a biometric template so that a portion of it is stored at one location, another portion at a different location, etc. Unfortunately, this approach does not lead to adequate protection: sensitive biometric data is still accessible to the servers and they compare templates in the clear (which leads to opportunities for abuse).

2.4 Smart Cards

Another possible mechanism for protecting sensitive biometric templates is to store them on smart cards that will control how the biometric template is accessed. Because the goal is to protect biometric data from the server, a smart card will need to be able to determine the outcome (e.g., recognition decision) without sending a user's biometric data to the server. This means that each user's template would need to be stored on a separate smart card, which makes this approach unsuitable for identification. Thus, we can only attempt to realize verification in this distributed setting.

A smart card is a physical card with an embedded integrated circuit that has limited storage and limited computational capabilities. It is powered by a smart card reader and interacts with it. Smart cards are designed to be tamper-resistant and use encryption to protect their memory content. They find uses in various domains, often providing authentication and other functionalities.

If we would like to employ a solution that relies on a smart card for biometric verification, the card issuer would first take a biometric reading and load the corresponding biometric template on a card. Then at the time of verification, the user would take a fresh biometric reading, which would be processed and communicated to the smart card to perform approximate matching with the template it stores. Conceptually, this accomplishes the goal of biometric-based verification, but falls short of providing critical properties.

The most profound drawback is that biometric templates are not protected from the entity issuing smart cards. That is, for a user to register for the service and obtain a smart card, the service provider needs to receive the user's biometric template and load it onto the user's smart card. This means that the service provider still

sees all biometric templates and is able to store them. Once again, this opens up opportunities for abuse of sensitive biometric data.

Besides privacy concerns, this solution also has significant usability drawbacks. First, a user's biometric template needs to be available at the physical location where smart cards are issued if we want to minimize the template's exposure. Second, using smart cards with biometric templates requires both a biometric reader and a smart card reader at the time of verification. This limits the applicability of this approach and does not make it suitable for mobile or desktop users.

3 Emerging Data Protection Mechanisms

Besides exploring conventional data protection mechanisms, we can also turn our attention to emerging technologies such as Trusted Execution Environments (TEE). A TEE is an isolated area of a CPU that allows us to securely execute applications on private data. This includes confidentiality and integrity protection of data used in the computation, and using only trusted software to work with the data.

Hardware isolation prevents other processes (including the operating system and other users) from observing the data placed inside the TEE. The TEE also comes with private keys embedded directly into the chip at manufacturing time. This private key material is likewise not accessible outside the TEE. Any software executed by the TEE must be certified using a key to which the TEE can build a chain of trust from a key loaded into it.

The notion of *chain of trust* is a common security concept that refers to a sequence of certificates from a public key that one trusts to a public key that needs to be verified. A *certificate* is a credential that permits one to verify the validity of the (public) key of an unfamiliar entity and use the key in security applications. In the simplest form, a certificate is a binding between the name of the entity and its public key, but often it contains additional data including the expiration date. The binding is achieved by means of a digital signature, which for an entity with name A and public key pk_A looks as follows:

$$\mathsf{data} = (A, \mathsf{pk}_A, \ldots), \sigma_{CA}(\mathsf{data}).$$

Here, let CA be a certification authority that we trust and let $(\mathsf{pk}_{CA}, \mathsf{sk}_{CA})$ be the authority's public-private key pair. The trust is typically achieved by pre-loading CA's public key pk_{CA} into one's computer. $\sigma_{CA}(\mathsf{data})$ is a cryptographic signature on input data produced using the private portion sk_{CA} of CA's key. A *signature scheme* is a cryptographic tool that uses public-key cryptography for ensuring data integrity. It guarantees a strong security property called *existential unforgeability*. Informally, a signature scheme is existentially unforgeable when it is infeasible for someone without access to the private key to produce a signature on any new message that has not been previously signed using that key. A signature is verified using the correspond-

ing public key. As before, "infeasible" means having at most negligible probability in the appropriate security parameter of succeeding in this task.

As a result, a certificate produced by an authority we trust provides validation of the data included in the certificate and extends the trust to another party. That is, if we believe that *CA* will not sign bogus data, we now have assurance that pk_A is the key associated with *A* and we can trust it. For example, if *A* is an online retailer, we can use pk_A to securely communicate payment information to *A* over the insecure internet.

In the context of the current discussion, the key that the TEE trusts is the signing key of a certification authority which was loaded into the TEE at manufacturing time. The key that needs to be verified is the key with which the software to be run by the TEE was signed. The path from the trusted key to the key we want to use can be longer than of length 1. In that case, we follow and verify a sequence of certificates. For example, if we trust key pk_{CA} and would like to verify key pk_A which *CA* did not sign, we need to find a sequence of certificates from *CA* to *A*, e.g., *CA* signed the key of *B* and *B* signed the key of *A*. This is why we refer to the concept as a chain of trust.

This process prevents TEEs from executing unauthorized software. In addition, any software that has been certified but subsequently altered will also fail verification and thus will fail to run. On the other hand, software that passes verification can be executed by the TEE and obtains access to the privileged information stored inside the TEE. Thus, this mechanism permits only trustworthy applications to access and handle data within the TEE.

When the application is not running, it can store private data outside the TEE in a protected form. The protection mechanism must guarantee both data confidentiality and integrity, which can be achieved by means of *authenticated encryption* designed to meet both security objectives at once.

We can imagine building an application that handles protected biometric templates for verification or identification purposes as follows:

1. When a biometric sample is captured (e.g., a fingerprint or face image), it will be processed and securely communicated to the TEE by trusted software run by the TEE.
2. The trusted software can use the key material embedded in the TEE to apply confidentiality and integrity protection to the biometric template and store it outside of the trusted environment.
3. Additional encrypted templates can be added to the protected storage over time when new individuals enroll.
4. When a new biometric sample is to be compared to one or more templates in the secure storage, the trusted software will load a protected reference template (from the secure storage) into the trusted environment, verify its integrity, decrypt it, and then compare the template from the new (probe) biometric sample to the stored (decrypted) template.

This conceptual realization of biometric verification or identification achieves proper protection of biometric templates. That is, both confidentiality and integrity of bio-

metric templates is achieved when they are stored outside of the TEE. Furthermore, when biometric templates are processed, all operations on cleartext biometric data are performed securely in the isolated environment using key material inaccessible to the operating system and users of the system.

While conceptually attractive, realizations of TEE-based solutions in practice require additional considerations and have certain drawbacks. Potential areas of concern include:

1. The solution requires hardware support and cannot be realized on systems where the processor, the motherboard, or the operating system does not support TEE, or on which the support is not enabled in the BIOS. Many computers are now capable of supporting a TEE.
2. The solution requires software attestation and certification. This is the most complex component of the process, because it requires proper management of various keys and ensuring that software being certified meets certain expectations (e.g., does not have vulnerabilities).
3. Most importantly, TEE is a relatively new technology and may not exhibit the level of maturity necessary for security applications. We can demonstrate that on the example of Intel SGX [1], which is a popular realization of the TEE architecture. A number of attacks and vulnerabilities on Intel SGX have been discovered that can lead to significant security violations (see, e.g., [2, 3]). One illustrative example is an attack on the chain of trust that allows for the recovery of a private key, which in turn enables execution of illegitimate software inside the SGX [7]. In the context of protecting biometric templates using TEE as described earlier, a successful attack of this kind would permit malicious software to obtain access to both stored biometric templates and incoming biometric samples, and perform arbitrary actions with them.

Thus, additional work needs to be conducted before TEE-based constructions can become a reliable solution for biometric template protection.

4 Conclusions

In this chapter, we discussed a couple of traditional cryptographic tools that are often used as data protection mechanisms. They include hash functions and standard encryption. We showed that it is challenging to use these tools by themselves to achieve adequate biometric template protection for the purposes of verification or identification. The main challenge stems from the misalignment of the properties of cryptographic tools and biometric matching mechanisms: cryptographic tools are designed to destroy any patterns in the data to which they are applied, while biometric data is inherently noisy and thus requires approximate matching. Other mechanisms that provide certain security properties, such as distributed databases and smart cards, are also not suitable for this purpose because they do not provide data confidentiality guarantees.

Trusted Execution Environments, an emerging technology, can be utilized to build a solution that supports secure storage of biometric templates and performs biometric matching by relying on hardware isolation. The main challenge here is that this is a relatively recent technology, which may not yet be widely adopted, may not have a desired level of maturity, and thus may be prone to attacks.

5 Summary

Traditional data protection mechanisms in the context of biometric template protection:

- **Hash functions** compute a fixed-length digest given an arbitrary-length input
  - hash functions have the one-way and collision resistance properties
  - hashes of two related biometric samples do not have any similarities
  - hashing individual elements of a biometric template does not achieve confidentiality
- **Encryption** provides strong confidentiality guarantees as long as the decryption key is private
  - ciphertexts cannot be meaningfully compared for biometric matching purposes
  - downgrading security to deterministic obfuscation still does not permit comparison of noisy biometric samples or templates
  - access to the decryption key during biometric matching nullifies protection offered by encryption
- **Distributed databases** and **smart cards** do not provide data confidentiality guarantees

Emerging data protection mechanisms:

- **Trusted Execution Environments (TEE)** rely on hardware isolation to achieve strong security properties
  - unprotected biometric templates reside only inside the TEE and are stored protected outside the TEE
  - decryption keys are used during biometric matching and are not accessible outside the TEE
  - TEE realizations have been a subject of powerful attacks

6 Study Questions

1. What purpose does a security parameter play in cryptographic constructions such as encryption schemes and hash functions?
2. Recall the simple mechanism for protecting stored user passwords described in Section 2.1. Now suppose that it is used for protecting the PIN number associated with a user's bank card. How many hash invocations would an attacker with access to the storage need to try to recover a user's 4-digit PIN stored in this form?
3. Suppose that someone uses deterministic encryption and sends a sequence of ciphertexts, where each ciphertext encrypts a single letter of the message. What can an adversary observing the sequence of transmitted ciphertexts deduce about the (plaintext) message?
4. Section 2.2 mentioned that block ciphers are designed in such a way that a single-bit difference in the input block (plaintext) impacts all bits of the produced output block (ciphertext). This is achieved by relying on the fact that block ciphers operate in rounds, where the difference gets amplified from one round to another. That is, a 1-bit difference in the input to round 1 results in a 2-bit difference in the output from that round; a 2-bit difference in the input to round 2 results in a 4-bit difference in the output from round 2; etc. What is the minimum number of rounds that a block cipher with n-bit blocks needs to have to guarantee that a 1-bit difference in the input plaintext impacts all n bits of the output ciphertext?
5. By going through your web browser's settings, can you determine how many certification authorities your browser recognizes?
6. What is the reason that a malicious user with access to the computer, the TEE of which performs privacy-preserving biometric matching, is unable to recover stored biometric templates?

References

1. Intel Sofware Guard Extensions (SGX). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html
2. Vulnerability CVE-2022-33196 (2023). https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-33196
3. Vulnerability CVE-2022-38090 (2023). https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-38090
4. National Institute of Standards and Technology: Advanced Encryption Standard (AES). FIPS 197 (2001). https://doi.org/10.6028/NIST.FIPS.197-upd1
5. National Institute of Standards and Technology: Secure Hash Standards (2015). https://doi.org/10.6028/NIST.FIPS.180-4
6. National Institute of Standards and Technology: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS 202 (2015). https://doi.org/10.6028/NIST.FIPS.202
7. van Schaik, S., Kwong, A., Genkin, D., Yarom, Y.: SGAxe: How SGX fails in practice. https://sgaxeattack.com/ (2020)