NIST Author Manuscript

NIST Author Manuscript

NIST Author Manuscript

Author Manuscript
Accepted for publication in a peer-reviewed journal

NIST National Institute of Standards and Technology • U.S. Department of Commerce

Published in final edited form as:
Computer (Long Beach Calif). 2018 January ; 51(1): 94–97. doi:10.1109/MC.2018.1151029.

Psst, Can You Keep a Secret?

Apostol Vassilev,
Research Team Lead in the Security Testing, Validation, and Measurement Group at NIST. He is also chair of the government–industry working group dedicated to modernizing the Cryptographic Module Validation Program through the adoption of advanced machine-based testing methodologies and automation

Nicky Mouha, and
Guest researcher in the Cryptographic Technology Group at NIST and an associate member of the CASCADE (Construction and Analysis of Systems for Confidentiality and Authenticity of Data and Entities) team of ENS (École normale supérieure) Paris

Luís Brandão
Guest researcher in the Cryptographic Technology Group at NIST

Abstract

The security of encrypted data depends not only on the theoretical properties of cryptographic primitives but also on the robustness of their implementations in software and hardware. Threshold cryptography introduces a computational paradigm that enables higher assurance for such implementations.

Protecting sensitive information from unauthorized disclosure has always been challenging. “Two may keep counsel, putting one away,” William Shakespeare wrote in *Romeo and Juliet* (1597). Later, in *Poor Richard’s Almanack* (1735), Benjamin Franklin wryly observed that “Three may keep a secret, if two of them are dead.”

Today, cryptography is a primary means of protecting digital information. In modern cryptography the algorithms are well-known; only the keys are secret. Thus, the effectiveness of encrypting data hinges on maintaining the keys’ secrecy. However, this is difficult in conventional cryptographic implementations, as keys are usually stored in one place on a single device, and used there to run the algorithm. This has led to the perception that cryptographic keys are often the Achilles’ heel of cryptography.

For example, the internal state of a conventional implementation might be compromised through a bug such as Heartbleed (https://nvd.nist.gov/vuln/detail/CVE-2014-0160), which lets an attacker read the application’s private memory, including any secret keys contained therein. Another example is the cold-boot attack,[1] which allows recovery of keys from DRAM even seconds to minutes after it has been removed from the device.

Disclaimer
The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

NIST Author Manuscript NIST Author Manuscript NIST Author Manuscript

Other attacks inject faults into the computation (for example, by changing the supply voltage), or obtain information through a side channel, such as the execution time, the amount of energy it consumes, or the electromagnetic emanations it produces. Many of these fall into the category of noninvasive attacks, which can be performed without direct physical contact with components within the device. Attacks that exploit leakage of key-dependent information can lead to disastrous scenarios in which the master key to encrypt and authenticate device firmware becomes compromised.[2]

To counter the inherent security risks of handling secret keys in conventional implementations of cryptographic algorithms, technical approaches have emerged that split the secret key into two or more shares. Each share independently processes data in such a way that the computation is correct as if the data had been processed by a classic algorithm with the original secret key. However, the compromise of one (or more, but not all) of the shares doesn't reveal information about the original key.

Splitting a key into shares combined with independent processing of the shares can significantly increase the confidentiality of secret keys in cryptographic implementations. However, it also presents challenges to ensure the correctness of the outputs that the user receives, and the continued availability of the overall system.

In this article, we focus on *threshold cryptography*: threshold schemes applied to cryptographic primitives and usually based on secret-sharing techniques. Threshold schemes also exist in other flavors, depending on the security aspects they address and the techniques used. They are related to the fields of secure multiparty computation, intrusion-tolerant protocols, and fault-tolerant and side-channel-resistant implementations.

Security considerations for cryptographic implementations

The basic security model for conventional cryptographic algorithms assumes an ideal black box, in which the cryptographic computations are correct and all internal states, including keys, are kept secret. Such ideal constructs have no side channels that could leak secret information. Under this assumption, one can reduce the problem of evaluating the algorithm's security properties to the complexity of the best-known attack against this model. For example, one can define the security strength, which can also be expressed as bit strength, of different classes of cryptographic algorithms based on the amount of work needed to perform a brute-force search of the key in a large space related to the key size.

When the algorithms are implemented in real hardware and software, the black-box assumption can break down in several ways. For example, bugs in the implementation can lead to side effects that compromise the secret key, as with Heartbleed. Also, the material and electromagnetic characteristics of the platforms on which the algorithms run can cause side-channel information to leak and allow attackers to recover the secret key.

The distinction of ideal versus real implementations can yield useful insights into the assessment of threshold schemes for cryptographic primitives. What are the security advantages and disadvantages of performing separate computations on shares of a key, compared to conventional implementations that use a single secret key? How can threshold

NIST Author Manuscript NIST Author Manuscript NIST Author Manuscript

cryptography mitigate the potentially disastrous consequences that a coding error or a side-channel leak could have on a conventional implementation?

Example threshold computation on secret shares

Secret sharing is based on splitting the key into multiple shares. For example, to split key K into three shares K_1, K_2, and K_3, we randomly select shares K_1 and K_2 from the same key space as K, and let the third share $K_3 = K_1 \oplus K_2 \oplus K$ be the one-time pad encryption of K, where $\oplus$ is the exclusive OR operation if the keys are bit-strings. No two shares provide any information about the secret key — all shares are required to recover K.

Now let's construct a threshold scheme for digital signatures. First, we recall the RSA (Rivest-Shamir-Adleman) signature scheme, which defines the public key as (N,e) and the private key as d, such that $ed = 1 \bmod \phi(N)$. Here, the modulus N is a product of two large secret primes and ϕ is Euler's totient function. Then, the RSA signature for a (possibly hashed) message m is defined as $s = m^d \bmod N$. Anyone possessing the public key can verify the signature by checking $s^e = m^{ed} = m \bmod N$.

To obtain a threshold variant of this signature scheme, we split the private key d into three shares d_1, d_2, and d_3, such that $d_1 + d_2 + d_3 = d \bmod \phi(N)$. Now, without reconstructing d, it's possible to first process the message independently using each of the shares: $s_1 = m^{d_1}$, $s_2 = m^{d_2}$ and $s_3 = m^{d_3}$; and then compute the signature $s = s_1 s_2 s_3$. Note that this is indeed a valid RSA signature, as $s_1 s_2 s_3 = m^{d_1+d_2+d_3} = m^d \bmod N$.

This simple threshold RSA signature scheme mitigates the risk of exposing the potentially high-value private key d, which doesn't appear in any of the three shares that are used in the actual computations. Thus, compromising any one of the shares, and even two of them, poses no threat of exposing d. Moreover, frequent updates to the key shares (d_1, d_2, and d_3) would reduce the window of opportunity for attacks and thereby further reduce the risk. The refresh can even occur after every signature.

For this scheme to work, all three shares must be present. This might be impractical in situations where one or more of the shares become unavailable. For such cases, a k-out-of-n threshold scheme could be used when at least k shares are available. Such secret-sharing schemes were independently developed by Adi Shamir and George Blakley in 1979. For RSA signatures, one can define a two-out-of-three secret-sharing scheme, and a corresponding threshold variant of RSA.[3]

Threshold cryptography against single points of failure

Conventional cryptographic implementations are susceptible to single points of failure, as shown by the Heartbleed and cold-boot attacks. But what do we gain by using threshold cryptography? For the example of a two-out-of-three threshold RSA signature scheme, consider the case of one share being irrecoverably lost or breached. Here the private signature key d remains intact, available, and not breached. This means that one can continue to use the same public key to verify the signature's correctness. In contrast, when a

NIST Author Manuscript NIST Author Manuscript NIST Author Manuscript

conventional implementation is breached, the corresponding public/private key pair would have to be revoked and a new pair issued, which typically requires an external certification of the public key by a certificate authority and propagating it to all relying parties.

In addition, a two-out-of-three threshold signature scheme becomes more resilient to future share loses if it continuously refreshes the key shares, provided that at most one is compromised at any given time. Interestingly, not all two-out-of-three threshold schemes are born alike. In a scheme composed of three separate conventional RSA implementations with independent keys, refreshing would require updating the public/private key pairs with all entailing inconveniences. Ensuring correctness might be more difficult for other cryptographic operations, such as encryption, but such issues have been addressed in the literature.

The secrecy of keys can also be compromised by the leaking of key-dependent information during computations. This is possible even without direct physical contact with components within the device. For example, the time taken, the power consumed, and the electromagnetic radiation emanated by a device can be measured without penetrating the device enclosure. In some cases, threshold cryptography can reformulate the side-channel leaks, making them more difficult or infeasible to exploit.

Consider, for example, an attack using power leakages, which requires obtaining traces of power across an algorithm's execution time. In differential power analysis (DPA), one collects power traces corresponding to a finite number p of statistical distributions of the power consumption, denoted as a p-th order attack. Security against p-th order DPA could be obtained if the attacker can recover at most $p < n$ shares — a $(p + 1)$-out-of-n threshold scheme would suffice.[4]

Under some reasonable assumptions on the statistical distributions of side-channel information, DPA requires collecting a number of traces that is exponential in the number of shares.[5] Therefore, the attack becomes infeasible if the number of shares is sufficiently high, and is further thwarted when the shares are refreshed before the attacker can collect enough traces.

Other attacks could inject a fault into the computation — for example, by applying a strong external electromagnetic field. If the threshold scheme doesn't require all shares to be present, it can resist transient and permanent faults in parts of the computation, thereby providing resistance against a wide range of fault attacks.

It might be that the threshold cryptographic implementation is insecure, perhaps due to a human error or an unsafe optimization by the tools used to compile or synthesize the implementation. It's important to ensure that the algorithms behind threshold cryptography are secure and well-analyzed, and to verify that they've been implemented correctly. These issues fall into the field of standardization and validation.

NIST Author Manuscript NIST Author Manuscript NIST Author Manuscript

Implications for standardization and validation

Governments recognize cryptography's important role in protecting sensitive information from unauthorized disclosure or modification and tend to select algorithms with well-established theoretical security properties. For example, US and Canadian federal agencies must use NIST-defined cryptographic algorithm standards to protect sensitive data in computer and telecommunications systems.[6] They must also use only validated cryptographic implementations, typically referred to as modules.

As we've pointed out in this article, the correct and bug-free implementation of a cryptographic algorithm and the environment in which it executes are also very important for security. To assess security aspects related to real hardware and software implementations, NIST established the Cryptographic Module Validation Program (CMVP; https://csrc.nist.gov/projects/cryptographic-module-validation-program) in 1995 to validate cryptographic modules against the security requirements in *Federal Information Processing Standard* (FIPS) *Publication 140-2*.[7] The CMVP leverages independent third-party testing laboratories to test commercial-off-the-shelf cryptographic modules supplied by industry vendors.

As technology progresses and cryptography becomes ubiquitous in the federal information infrastructure, the number and complexity of modules to be validated increases. This makes it increasingly difficult to detect at validation stage all possible defects that might compromise security. This is one more reason to consider the potential of threshold cryptography in avoiding single points of failure in real implementations. The definition of guidelines would help to develop a structured process of formulating and validating security assertions about threshold cryptographic implementations. One additional challenge is to enable ways to validate those assertions in an automated fashion.

As the use of cryptographic algorithms increases, threshold cryptography becomes increasingly relevant in addressing obstacles arising from differences between ideal and real implementations. A major issue is the protection of secret keys, which cryptographic security relies on. Threshold cryptography enables secure modes of operation even when some components are compromised, but they also present new challenges for the standardization and validation of security assertions.

References

1. Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, Feldman AJ, Appelbaum J, Felten EW. Lest We Remember: Cold Boot Attacks on Encryption Keys. Proc 17th USENIX Security Symp, (USENIX Security 08). 2008:45–60.
2. Ronen E, Shamir A, Weingarten A, O'Flynn C. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. Proc 2017 IEEE Symp Security and Privacy (SP 17). 2017:195–212.
3. Shoup, V. Practical Threshold Signatures. In: Preneel, B., editor. Advances in Cryptology—EUROCRYPT 00. Springer; 2000. p. 207-220.LNCS 1807
4. Nikova, S., Rechberger, C., Rijmen, V. Threshold Implementations against Side-Channel Attacks and Glitches. In: Ning, P.Qing, S., Li, N., editors. Int'l Conf Information and Communications Security (ICICS 06). Springer; 2006. p. 529-545.LNCS 4307

NIST Author Manuscript

NIST Author Manuscript

NIST Author Manuscript

5. Chari, S., Jutla, CS., Rao, JR., Rohatgi, P. Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M., editor. Advances in Cryptology—CRYPTO 99. Springer; 1999. p. 398-412.LNCS 1666

6. Information Technology Management Reform Act of 1996. Public Law 104–106, section 5131. https://www.dol.gov/ocfo/media/regs/ITMRA.pdf

7. Federal Information Processing Standards Publication 140-2. Security Requirements for Cryptographic Modules, NIST. 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf