Introduction to SplitKey Foundations White paper

December 12, 2018

10 pages

Contents

1 Introduction

Document explains the mathematical formulas and principles behind the SplitKey technology. The document includes information security notions, intended to assist the security evaluation of the SplitKey Platform.

1.1 Purpose

SplitKey is a technology bringing true cryptographic *authentication* and *signing* to the commodity platforms of smart devices (iOS, Android), while not scarifying the security levels characteristic to specialised hardware platforms, such as hardware smart-cards or hardware security modules. SplitKey can be thought of as a mixed token solution for authentication and signing, combining the benefits of both hardware cryptographic token (security wise) and soft token (regarding the convenience).

The document is describing SplitKey in action. The internals of the SplitKey technology are explained in fine detail. Some basic knowledge of information security, legislation and cryptography is advised. The paper also addresses some security issues that SplitKey is able to solve. The mathematical apparatus behind SplitKey has been originally noted in [1].

1.2 Participating Actors

There are two participating Actors. The first one is the *Client* who initiates operations using a commodity Smart Device. According to the laws and standards, the Client is the owner and a legal sole possessor of the full, final key. In case of SplitKey, the Client does not have to blindly trust the infrastructure, but is in real control over his key.

The other Actor is the infrastructure of the Service Provider. For clarity reasons, the current paper abstracts from the actual complexity of systems available at the Service side and refers these jointly just as the *Server*.

The important role of the Server side is to enforce high level technologies like HSM and industry standard procedures to support the Client in exercising his sole control over his private key. It is the task for the Server side to establish the assurance, trust and auditability due to the inability of a commodity Smart Device to achieve these goals in isolation.

1.3 Cryptography Foundations

SplitKey could be seen as a superstructure to the standard RSA algorithm [2]. The full final (private) key d is an implied compound key of multi-prime nature. However, according to an important design goal of the SplitKey, constituents for the virtual key d are never explicitly gathered together. The full private key d currently has the length of 6 Kibit (6144 bit).

The virtual private key d can (for clarity reasons) be depicted as having a vertical cut at the splitting point. In reality, this key is built bottom up — its half-keys are generated by distinct Parties.

The first half-key d_1 is to be generated by the Client while the second half-key d_2 is to be generated by the Server, as illustrated on Figure 1. Both keys are of 3 Kibit (3072 bit) length.

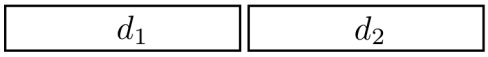

Figure 1. The first *split*, actually a compound

Instead of ever putting these two half-keys together, the meaningful key operations are conducted in separation — i.e. sequentially with both half-keys. Due to the multi-prime RSA math, the result is an equivalent to the operations with the full final key.

SplitKey technology yet provides a second, technologically more complex split applied to the half-key d_1. This is a real split based on addition modulo the Euler's totient function. The second

split could be better described as a *horisontal* one, see Figure 2, due to the fact that resulting keys d'_1 and d''_1 both are of 3 Kibit size.

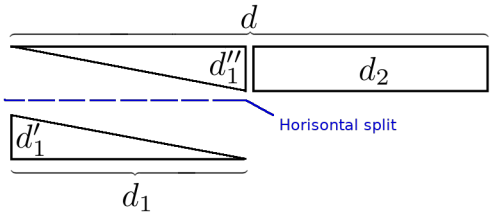

Figure 2. The second split

Summarizing — the full private key d is virtual and as depicted on Figure 3, can be thought as of a compound of parts d'_1, d''_1 and d_2.

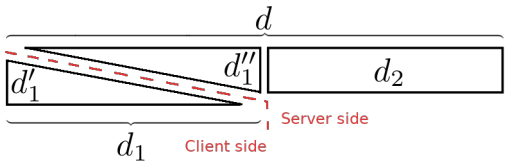

Figure 3. Both splits with storage locations marked

1.4 References

[1] A. Buldas, A. Kalu, P. Laud, and M. Oruaas, "Server-Supported RSA Signatures for Mobile Devices", in *Computer Security – ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, pp. 315–333, isbn: 978-3-319-66402-6. [Online]. Available: https://doi.org/10.1007/978-3-319-66402-6_19 (visited on 09/03/2018).

[2] *PKCS #1: RSA Cryptography Specifications Version 2.2*, RFC 8017 (Informational), IETF, Nov. 2016. [Online]. Available: https://tools.ietf.org/html/rfc8017.

[3] *Common Criteria for Information Technology Security Evaluation. Part 1: Introduction and general model*, Apr. 2017. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf.

[4] *Common Criteria for Information Technology Security Evaluation. Part 2: Functional security components*, Apr. 2017. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R5.pdf.

[5] *Common Criteria for Information Technology Security Evaluation. Part 3: Assurance security components*, Apr. 2017. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf.

[6] *Trustworthy Systems Supporting Server Signing. Part 1: General System Rrequirements*, draft prEN 419 241-1:2017, version 0.15, Oct. 2017.

[7] *Trustworthy Systems Supporting Server Signing. Part 2: Protection Profile for QSCD for Server Signing*, draft prEN 419 241-2:2017, version 0.15, Oct. 2017.

[8] *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC*, Aug. 2014. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv%3AOJ.L_.2014.257.01.0073.01.ENG.

2 Key Generation Phase

A key generation procedure will always be initiated by the Client and never by the infrastructure. The Client shall be the future owner and sole legal possessor of the full, virtual final key.

2.1 A static overview of the involved elements

In this section, the meaningful objects are enumerated and the algorithm used to split the (private) half-key d_1 is explained, which obviously is the most interesting object throughout the SplitKey framework.

The effective (or combined, or final, or full) RSA private key of the Client shall be d. As a paradox, that key never participates in cryptographic operations, neither does it exist in an evident form, thus it is not possible for an attacker to directly obtain the key d. The functions with d are supported indirectly, via mathematical operations.

The Client part d_1 of the final RSA key is generated at the Client side. Server part of the key d_2 is generated at the infrastructure side. The key d_2 stays at the Server side, thus it cannot be obtained by attacking the Client side. As the result, to obtain the full private key, attackers would be expected to compromise both the Client and Server sides, which is a highly unlikely event.

Along with the private keys, public keys are also generated — according to RSA requirements. The final (full, compound) public key of the client as later referenced by the Client Certificate, shall be n .

The notion of public keys is similar - the Server part of the public key is referenced as n_2 and it is created by the infrastructure part. The Client part of the (final, public) key n is called n_1 and it is prepared by the client.

Admitting the pre-defined exponent e is important for the underlying mathematics, it is not actually transferred during the actual key operations. With that simplification, the full public key $n = p \cdot q$ is respectively comprised from two half-keys $n_1 = p_1 \cdot q_1$ and $n_2 = p_2 \cdot q_2$. The full public key appears as the multiplication of four primes: $n = p_1 \cdot q_1 \cdot p_2 \cdot q_2$.

A distinctive property of SplitKey is the transformation chosen to yet further split the Client part of the Client private key, following its initial generation. With that purpose, some extra operations are applied on the private key d_1 (albeit not on its public counterpart n_1).

The reason to split the key d_1 is to *divide the key between two parties* so that d_1' will be stored at the Client side and d_1'' at the Server side. The particle d_1' is stored within a relatively hostile environment of the Client smart device, it must be virtually indistinguishable from a random string. SplitKey makes use of *additive sharing* for both purposes.

Lets use a simplified mathematical notation[1], marking the split $d_1 = d'_1 \oplus d''_1$. According to that notion, the key d'_1 can be calculated as $d'_1 = d_1 \ominus d''_1$.

First, a random value of a pre-defined size (3 Kibit) is requested from a random number generator, to be further used as d''_1. The obtained bit string obviously possesses the random qualities the private key d_1 is lacking. The task solved by the operation of additive sharing, is dual: to split the newly generated key d_1 into two pieces while *imprinting randomness* on a newly formed particle d'_1. The chosen way of splitting will result in two effectively random bit strings which, if combined, still retain certain mathematical properties in respect to RSA.

At the Client side, extra measures are undertaken to conceal the value of d'_1 from the attackers so it will be stored encrypted by AES. Consequently, a PIN will be later needed to *open* that key.

The d''_1 part is to be sent to the Server and stored in HSM. The modulo sum $d'_1 \oplus d''_1 = d_1$ is true while both addends remain effectively random and are stored at distinct parties, thus the attacker's chances are extremely complicated. To compromise the private half-key d_1 demands compromising both sides, which could be considered extremely unlikely due to the mitigation efforts provisioned by design, built in and audited at the infrastructure party.

Last but not least, a document or transaction to be signed should be determined prior to signing. Due to the known limitations of the RSA cryptosystem (speed, requirements), the actual input for the signature operation will be the message m obtained by first by hashing the document D (e.g. by applying a function outputting a 256 bit sequence) and then padding the hash value to be commensurate with the 6 Kibit key intended to sign with.

This was an overview of cryptographic elements in static, as well as the explanation of their purpose. Subsequently, a dynamic overview will be conducted explaining the basic operations with the elements.

2.2 Key generation: a dynamic notation

For legal implementation, the keys do not exist in isolation but are used together with the personal certificates. According to the standards, a certificate is formed from the public key of the owner and some extra data, by signing the outcome cryptographically by someone else (normally a CA).

Generating a new certificate includes several phases, the actual generation of the keys, splitting those according to the SplitKey technology requirements, and the final operation of certification resulting in the fresh public key published at a CA. The last operation is a technological one — erasing the sensitive cryptography material used during the key generation but not needed after that.

A disclaimer: cryptographic operations are described below in an order easy to understand which is not necessarily the exact sequence the software is internally using for the purpose.

2.2.1 Key Generation

Generating new keys is an operation of paramount importance — because a new identity is created. Thus, the Client is expected to authenticate prior to each attempt. Here is the workflow:

- The *Client* initiates the key-generation via GUI.
- A standard RSA key generation operation is launched to obtain a public key n_1 together with a paired secret key d_1, both of 3 Kibit size;
- The Client will transfer the public part n_1 of the newly generated key to the Server.

[1] The actual cryptographic operations $\oplus$, $\ominus$ are the addition and subtraction modulo the Euler's totient function $\varphi = (p_1 - 1)(q_1 - 1)$

- In parallel to that, the *Server* launches an independent RSA key generation task to obtain the keypair n_2 and d_2, both of 3 Kibit size.

At the Server side, the keys are stored within an HSM.

2.2.2 Splitting the Private Key

- The Client will recalculate d_1 as two equivalent constituents d_1' and d_1'' according to the operation of additive sharing (see section 2.1 A static overview of the involved elements);
- the constituent d_1' will be stored by the Client at its file system (encrypted, more about that later);
- the constituent d_1'' will be transferred from the Client to the Server, via a secure channel;
- the transferred constituent d_1'' will be marked for a later erasure;
- the initial half-key d_1 is marked for a later erasure.

The result of the splitting will manifest a mathematical meaning later, during the signing operations. Until then, the constituents d_1' and d_1'' look like random bit strings.

2.2.3 Certification

To onboard the newly generated key pair into the trusted system, the key pair first has to be bind to the user's identity. The outcome from the certification phase is a new identity — a certificate within the Service Provider's system. A suitable graphic meme for a key pair with the certification seal applied is presented on Figure 4.

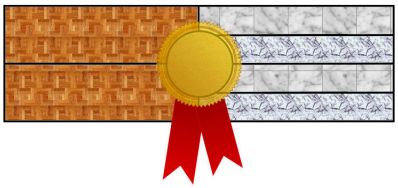

Figure 4. The arrangement of a compound key pair depicted as a domino piece with a certifying seal on top

The procedure runs as a continuation to the key generation operation. This is the procedure:

- The server prepares parts needed for the certification, such as the public subkey n_1 obtained from the Client and its own subkey n_2.
- Server forms a *signature activation request*. Based on the identity of the authenticated person (obtained via other methods), a CSR (Certificate Signing Request) is initiated, containing:
  - a (possibly qualified) identity of the person as advised/derived by an external *Registration Authority* or by other methods;
  - the compound public key $n = n_1 \cdot n_2$;
  - a timestamp (satisfying the legislation);
  - definition of the certification policy;
  - URL linking the CRL;
- Provided all the formal requirements were honoured, the server shall contact the CA and submit CSR and apply for certificate.

- The expected result is a fresh personal certificate (of a new identity) returned from CA and securely stored by the Server.
- The Server produces relevant audit logs about an event.

2.2.4 Erasure of remnants

On successful ending of the key generation operation:

- The Client will securely wipe the primes p_1 and q_1 ;
- The Client will securely wipe the key d_1 meant to be never directly used;
- The Client will securely wipe the key d_1'' previously transmitted to the Server. This is because only the Server should be in control of that part.

If the certification request did not reach to the intended outcome, then all the generated cryptographic material becomes worthless and is to be destroyed — advisably in a secure fashion.

3 Signing phase

As demonstrated above, the full final private key is now split between the Client and the assisting infrastructure. The usefulness of the SplitKey manifests itself in a way that both sides will sign the message m in a full separation from each other and then clever mathematics is used to form a full signature out of the partial results.

Taking into account that the Server can neither initiate the signing operation nor finish it due to the lacking d_1' particle, the situation could be reduced to the *Client in sole possession of its private key*, which is an important goal from the viepoint of [3]–[8] while the infrastructure is only assisting the Client with the procedure quality otherwise unachievable by mobile devices.

3.1 Preparing the input

Relying Party (RP) in legal agreement with the Service Provider, can offer documents or transactions to be signed by the Client. RP is the one in possession of the original document/transaction.

The following will take place: the RP in possession of the original document, will calculate the hash (message digest) H of it and will pass that value to the Server (it takes place out-of-band of the Client-Server communication channel). The Server will subsequently pad the digest H forming a message m to be signed.

- The RP shall calculate the hash (message digest) H from the attempted transaction and pass it to the Server.
- The Server will pad the digest H forming the message m to be signed. The size of the padded message is commensurable with the full 6 Kibit key size (a requirement derived from RSA internals).
- The Server will communicate the padded message m to the Client.

3.2 Signing a message

The current procedure assumes the document or transaction to be signed is originating from an RP — relying party, e.g. a bank or service portal. Technically a variation is possible with the document to be signed located e.g. at the Client file system.

3.3 Creation of subsignatures

The Client and Server will independently proceed with the calculations of the respective signatures. Either is only able to calculate a partial signature based on the subkeys d_1' and d_1'' in their respective possession.

3.3.1 Client part of the Split

- The Client attempts to decrypt the key d_1' — a PIN is needed for this operation. The beauty of the mathematics ensures the AES encrypted value will be *opened* with any PIN, including the wrong ones, thus the attacker has no feedback on the success.
- The Client takes the message m to be signed and calculates the partial signature according to the expression $m^{d_1'} \bmod n_1$ (which is a standard RSA signature creation formula). If the PIN was incorrect, then the key d_1' was incorrect, too. If so, then the subsequent operations are not meaningful, albeit they could be technically considered successful.

3.3.2 Server part of the Split

- The Server obtains the respective subkey d_1'' securely stored at the HSM and only usable at the client's request. Any use of that key is strictly logged and audited.
- The Server takes the message m to be signed and calculates the second partial signature according to the expression $m^{d_1''} \bmod n_1$.

3.4 Compilation of the signature from Splits

The mathematics behind it, how seemingly random bit strings form a complete signature, is:

$$\begin{cases} m^{d_1'} \bmod n_1 \\ m^{d_1''} \bmod n_1 \end{cases} \Leftrightarrow m^{d_1'} \cdot m^{d_1''} \bmod n_1 = m^{d_1'+d_1''} \bmod n_1 = m^{d_1} \bmod n_1.$$

As described above, the mathematics on the Client side works transparently regarding the PIN value. An attacker on the Client side has no means to check the correctness of the formed subsignature. However, the server, now possessing both the subsignatures, is able to form a half signature and apply the main equation of RSA

$$\left(m^{d_1}\right)^e \bmod n_1 = m.$$

This way, the Server is now able to indirectly verify that the Client PIN was correct and consequently to prove that *the Client was in sole control of its secret key*, as requested by [6] and [7].

3.5 Forming the Full, Final Signature

Only if the previous check was successful (and the sole control of the private key was explicitly demonstrated), the Server considers it reasonable to calculate another half of the full signature. The second half of the signature could be regarded as the server's consent and assurance about it, that the derivation of the first half-signature was produced according to the full ruleset, any applicable controls and standards. The actual formula used to calculate the second half-signature, is:

$$m^{d_2} \bmod n_2$$

The output of that operation is considered to be always correct, thus extra checks do not apply — the second half-signature is needed for nominal reasons to form a full signature. The operation for compiling the full signature is:

- The server applies the Chinese reminder theorem to the *concatenated* half-signatures $m^{d_1} \bmod n_1$ and $m^{d_2} \bmod n_2$ to form a full signature $m^d \bmod n$.

It is worth remembering the full private key d never existed and was never stored. This is a significant fact due to the necessity to prove the sole control over the private key, which can now be fully attributed to the Client. Vice versa, storing the full key at the server would compromise the requirement of sole control.

3.6 Verification of the Full Signature

The operation of public verification of the signature is possible because the public key n is published as a part of the personal certificate available via PKI. Thereby the operation is accessible not only to the infrastructure Server but to any party having access to the signed document and the public key n of the Client.

3.7 Extra items

Two important extensions are still to be kept in mind:

- In contemporary systems, there is a tendency to generate a set of two key pairs — one for authentication purposes, another strictly for signature operations. This effectively doubles the extent of complexity.
- The legal digital signature is normally not the same as technical signature generated with cryptography. To obtain the legal strength according to reg. (EU) 910/2014 [8] provisions, a timestamp has to be present and an approved relation demonstrated with a legally accepted CA.

3.8 Summary

The mandatory operations involving SplitKey were extensively discussed above to the extent necessary to understand the specific properties of the SplitKey technology. Certain generic security threats were discussed to demonstrate their relation to the SplitKey system.