The Basics of Multi Signatures using RSA

Jonas B. Erasmus, Dr. Holger Kinkelin*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: jonas.erasmus@tum.de, kinkelin@net.in.tum.de

***Abstract*—Requiring multiple parties to generate a single signature can improve security and accountability. This paper presents a scheme for Certificate Authorities to have two signers generate and verify a single certificate, without the client noticing. Additionally shortcomings and alternatives to multi key signatures are explored.**

***Index Terms*—Digital signature, Multi-Signature, RSA, Public Key Cryptography, Certificate Authority, Secret Sharing**

1. Introduction

In the internet almost all traffic is authenticated, the sender signs the message to prove it was sent by them. While this system works efficiently on a personal basis, larger entities or companies also sign messages using only one key. As a result, whoever knows that key may authenticate messages. While this is a security risk, additionally the question arises if a single person should be able to sign in the name of the whole company. Certificate Authorities for example sign the certificates of websites. Even with the whole verification process for a domain running successfully, in the end a single key (and as such a single person) signs the certificate. The need to verify messages is without question. But with a single signing key, whoever is in control of this key can officially speak for the whole company. While this is an issue of trust, there also is the risk of human error. And even if a misuse is noticed, the tracing of who used the signing key for what grows ever more complicated with the size of the company. This paper presents a solution of multiple keys being used to generate a single signature. This can enforce multiple signers checking the signature and only if every participant is satisfied a valid signature is generated. Additionally none of the signers is in possession of the master key, but only knows a key part.
First the basic concept of signing is explained using RSA. Then Colin Boyd's suggestion of an dual signer approach [1] is presented, and applied to certificate issuing. Finally the shortcomings and alternatives to multi key signatures are explored and evaluated.

2. Fundamentals

For the scope of this paper, only the signing of messages using RSA (Rivest–Shamir–Adleman) will be investigated. As such, the main focus is on the keys used by the signing entity. For basic operation only two keys are required. Key *e* is the public key to be shared with all other users, while *d* represents the private key that is kept hidden from other entities. For the sake of simplicity key generation and distribution is omitted.

2.1. RSA

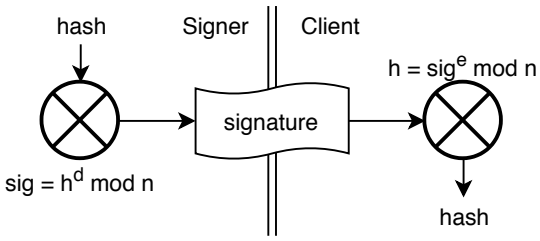

Figure 1: Basic RSA signing scheme

The RSA cipher (as depicted in Fig.1) is widely used in today's internet to sign and verify the authenticity of messages. The to be signed message *M* is hashed to a hash-value *h*.

$$h_s = hash(M) \tag{1}$$

Subsection 5.2 of the corresponding RFC [2] (Request-for-Comments; Part of the Internet standardization) describes the algorithm as follows. With the given *h* as the hashed message, and the key *(d, n)* (private key, and publicly known modulus) a signature *sig* may be computed.

$$sig = h_s^d \mod n \tag{2}$$

For further use we will refer to signing a Message *M* with Key *d* using RSA as *RSA(M, d)*. The signature is then appended to the message and the finished package is sent to the receiver. Here the message can be verified using the known public key *e* of the signer.

$$h_r = sig^e \mod n \tag{3}$$

$$h_r \stackrel{?}{=} hash(M) \tag{4}$$

Should (4) hold true, the receiver knows, that the message was indeed sent by the signer. As an imposter could not generate the correct signature for their message (since they cannot know the required private key).
In a minimal notation RSA signing can be written as

$$sig = RSA(M, d) \tag{5}$$

for signing, and

$$hash(M) \stackrel{?}{=} RSA(sig, e) \tag{6}$$

for verification of a signature.

doi: 10.2313/NET-2019-10-1_02

2.2. Important RSA Properties

The usage of multiple keys in the RSA algorithm is straight forward. A signature may be signed a second time, however the result is the same as if signing with a "concatenated" key (explained in depth in Sec.3.2).

$$RSA(RSA(M, d1), d2) = RSA(M, d1.d2) \quad (7)$$

Additionally RSA is commutative, the key order for multi-signing is irrelevant.

$$RSA(RSA(M, d1), d2) = RSA(RSA(M, d2), d1) \quad (8)$$

By using this property for two random keys *d1* and *d2*, a third public key *e* can be found fulfilling the public/private key properties

$$hash(M) = RSA(RSA(M, d1.d2), e) \quad (9)$$

As such *d1* and *d2* can be interpreted as two parts of a private key. When choosing n ($n > 1; n \in \mathbb{N}$) random keys, an inverse key (that is a key to decrypt the message) can always be found.
The security of the use of multiple RSA steps is still enforced. Without knowing the master key *d1.d2*, it is not possible to derive *d1* when only knowing *d2* and *e* (or vice versa). The splitting is only defeated once multiple key parts of the private key are known to the same entity. It could then compute the concatenation of those keys, obtaining a larger key part of the master key (or the master key itself if knowing all key parts).

3. Multi Key RSA Signing

This section describes how the RSA algorithm can be extended to allow multiple parties to generate a single signature.

3.1. Goal

The important part of multi key signing is that a client should be unable to tell a difference to a "single-signed" signature. As such the signer may use a multi key scheme, however there is still only one public key and the receiver (client) can verify the signature using the common RSA verification. In order to do so the private key is split into multiple keys, however the public key only verifies the concatenation of all private key parts.
Additionally the signers may not be able to generate a valid signature on their own. Multiple signers need to sign in order for the signature to be correct. Still a central authority, rolling out the individual key parts, is needed.

3.2. Key Distribution

In order to construct all the partial keys, full information of all involved keys is needed. To derive the partial keys the *master key* (the 'un-spilt' private key) needs to be known to compute the corresponding *public key*. As such a central authority needs to compute all the partial keys *d1* to *dn* and the public key *e*.
By using the standard key generation algorithm so called 'backdoor' information is generated. Using this extra knowledge an inverse key can be computed. It is also possible to find the inverse of multiple concatenated keys (by treating the product over all private keys as the master key). While computation time is higher than normal (the possibly large keys need to be multiplied) key generation only needs to be done once.
The partial keys then need to be distributed (via a secure channel) to all signers participating in the process.
The following explanations assume the partial keys were successfully distributed and the public key *e* is well known by all entities. The modulus *n* is a constant (all keys and partial keys are of the same "length") and as such is known, or even hard-coded into the RSA algorithm.

3.3. The Basic Idea

This subsection is a recollection of Boyd's suggestion [1] for two signers as depicted in Fig.2.

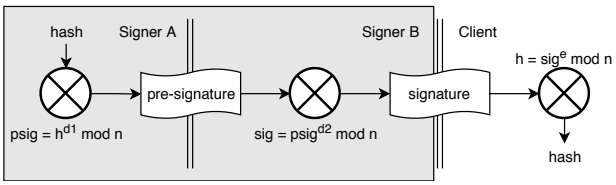

Figure 2: Signing using two Signers

With two signing parties, three keys are needed. The private key is split into two parts *d1* and *d2*, and the public key *e*.
The first signer signs the message using its own (secret) key part *d1* generating an incomplete signature.

$$psig = RSA(M, d1) \quad (10)$$

The second signer can now create the final signature using their own private key (11). This signature should not be published, unless (12) holds true (the second signer checks the correctness of $psig$).

$$sig = RSA(psig, d2) \quad (11)$$

$$hash(M) \stackrel{?}{=} RSA(sig, e) = RSA(psig, d2.e) \quad (12)$$

For a client this process is transparent, the public key *e* is the only information needed to verify the final signature. Since RSA is commutative (Sec.2.2 Eq.(8)), the order of signing is not important.

3.4. Problems when Extending to More Keys

To include more signing parties, more keys need to be generated. For example three parties need three keys (*d1,d2,d3*) and one public key *e*.
The first signature can be generated as in Sec.3.3.

$$psig1 = RSA(M, d1) \quad (13)$$

The second signer still needs to verify the signature (the first could have lied, and signed a manipulated message). The only knowledge is *psig1*, as well as the local secret *d2* and the public key *e*. However at this stage the unknown key *d3* of the third signer is needed for verification:

$$hash(M) \stackrel{?}{=} RSA(psig1, d2.d3.e) \quad (14)$$

doi: 10.2313/NET-2019-10-1_02

For more signing parties, even more secrete keys of other participants are needed. Without verification a signature should not be signed, since its integrity cannot be proven. With this "blind" signing, only the first and last signer can verify the signature. All intermediate signers can not contribute to the correctness (or truthfulness) of the signature.
Another alternative would be each signer adding their own signature to the document. However this defies the requirements of Sec.3, a client should only need to verify a single signature with a single public key. Additionally, for larger numbers of n signers a large overhead in signature length (n times more signatures) is generated. Needless to say that the verifier needs to check all n signatures, requiring the trusted knowledge of all n public keys.

3.5. Multi-Signature Schemes

An often presented solution are identity-based multi-signature systems. Here the signature is generated by using the identities of the signing parties. While this is an important solution to decrease the amount of public keys each participant needs to know (while maintaining security) and the complexion of signature verification [3] [4], these systems are not efficient at solving the problem of this paper. Especially since the verification process needs to be adapted to the signing algorithm.
However "A review of multisignatures based on RSA" [5] presents some schemes that could be adapted to fit the use case we need. However most of the presented ciphers require special prerequisites. As presented in [6], key generation is possible without any participant knowing the master key.

4. Example of Multi Signature for Certificate Authorities

In this section the system of Sec.3.3 is applied to a dual signature scheme for Certificate Authorities.
For sake of simplicity, a *website* will be *signed* resulting in a *certificate*. The website includes the full request for a certificate (used public key, website-operator, contact information, etc.).
Certificate Authorities (CAs) are tasked with verifying websites and signing their certificate. However the CAs use a single key pair to sign a record. With multi-key ciphers (and multiple signing keys) the risks of miss signings, or unauthorised signings could be reduced. It is assumed a website has been cleared for signing. Under normal circumstances, the CAs signing key would be used to generate the certificate. This can result in a number of problems. The signature will be secure, however the website might not have been cleared for signing.These errors are induced by the staff at the CA. And the easiest way to reduce these risks, is to establish a second (or even more) pair of eyes. A miss signing of one employee may be noticed by a co-worker. Additionally the responsibility (and the trust) for a certificate is evenly spread amongst several employees of the CA. A better way to handle the final signing of the website's certificate would be as follows (see Fig.3)

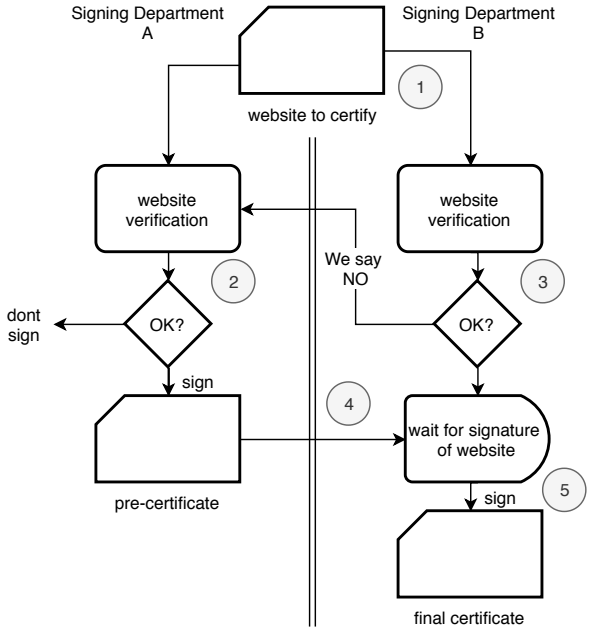

Figure 3: Basic RSA multi-signature scheme for CA

1) The website information (public key, owner, domain, etc.) is handed to two different departments. For sake of simplicity, one party is called *primary* signer. However, the signing order is irrelevant. Both departments only know one (different) key part of the master signing key.
2) The website is verified by the signers. This is done as if there was no second party involved. As a result this step can be taken in parallel: This will increase workload, but issuing time for a certificate remains the same (as with one signer).
3) Should the verification fail, the other department needs to be notified, in order to stop the signing process. Additionally the certificate will obviously not be signed with the local key.
4) If the primary signer trusts the website, it may issue a pre-certificate (using the secret key part) and send it to the second department.
5) Since (for two signers) the second party can fully decode the pre-certificate, it can be "re-verified". Due to working in parallel, at this point the second department probably is in the process of verifying said website their self (between stages 2 and 3). Should the website pass, signing the pre-certificate results in the final certificate.

In the given example (Fig.3) the primary signer will finish verification before the other department. However there are only two steps, where information is exchanged. On the one hand the website could not be verified (step 3)). In this case the first department to fail will contact the other one and no certificate is issued. Or everything is fine and one party has issued the pre-certificate. In step 4) the signer should first check if there is already a pre-certificate of the website, if there is they were slower than the other depratment and need to skip to step 5).
In this "two person signing" scheme, the two involved departments must have been supplied with the two different key parts. This needs to be done every time a new

doi: 10.2313/NET-2019-10-1_02

signing key is generated. Effectively the company is split into two parts, both need to work together to issue a single certificate. Spreading this further two CAs could work together to generate a "two CA trusted certificate". With intelligent use of multiple keys spread across CAs, this system would distribute trust to more than one company for each certificate.

5. Secret Sharing

The most used system of secret sharing (and the most used scheme) was invented by Adi Shamir in 1979 [7]. The basic idea is that a secret (in our case the private master key) is split into multiple parts. In [8] the system of splitting a key into multiple parts is already explored. Although this is not a multi key cipher, the possibility to split (or share) a key secretly is of fundamental need for multi-key ciphers. But instead of using the partial keys in a multi key cipher, other options are also possible.
The master key can be split (by simply using XOR operations) and the parts distributed. A central trusted entity then collects these partial keys, reconstructs the "master key" and signs the document. Obviously the central authority is in possession of the master key (or can read the key when all partial keys are entered). While this approach is heavily utilised for secure key storage [8], the partial knowledge forces the signers to rely on a central entity for signing. If all participants are part of the same company this might even be wished for, and there is already a "trusted" master entity (e.g. the boss, or board of directors). Compared to multi-key schemes this approach lacks security, since it returns to a single entity, that may not be compromised, and reintroduces the problem of "trust".
By using the properties of secret multi party computation, the knowledge of the master key can be circumvented. [9] describes a program, that computes a (public) RSA signature using the input of secret master key parts. This approach bridges the gap of limited trust and good usability. Here the central trusted entity is replaced by cryptographic primitives. All participants compute a RSA signature only with their local knowledge, by exchanging additional information (that carries no readable secrets), the final product will be the correct signature (with no one actually knowing more than before).
A new opportunity of strongly integrating secret sharing are (t,n) threshold schemes [10]. Instead of all n (n should be > 2) entities with a partial key, only a certain subgroup of $t < n$ are needed to successfully compute a solution. While this may not be a requirement for signatures, it increases the versatility of a signature scheme. For example multiple departments of a company can generate a signature. With this scheme, only 4 (out of 8) are needed for a valid signature. This reduces workload (only 4 departments must verify a website) and any 4 departments can generate a signature together (making load balancing across all work groups easier).
A possible danger of secret sharing are dishonest parties. By sharing faked information, many secret schemes may be defeated (the system won't generate the desired result and even secret information may leak). To ensure that all participants are using the correct algorithms and do not try to manipulate the scheme, verifiable secret sharing (VSS) [11] is needed. These algorithms enforce a correct utilisation of the scheme for all participants. If not trusting another participant, the integrity of their computation needs to be enforced this way.

6. Conclusion

The implementations of multi-signature systems are complicated. While they are feasible and can be used, the setup for signing a document with multiple signers is too difficult for the gain. As detailed in Sec.3.3, two signers can easily collaborate in creating a valid signature. Approaches requiring more participants are often required in a regulated environment. It is fair to assume that all entities are part of the same organisation, or at least know each other and have secure means of communication. In other words a trusted third party or centralised authority can be established (in fact [1] and schemes from [5] require such an authority). At this point the authority can also wait until all signers verified the document and then sign using a single master key.
The approach of incorporating secret sharing algorithms can bring back cryptographic security, while still spreading the signature process over multiple signers. Still such schemes are not straight forward and require trust or strong (and complex) cryptographic primitives.
In the end, the trade-off between risk of missignings (as well as security) and the requirements for implementing these systems needs to be considered. As long as all signing is done within the same entity, policies and management might be able to enforce better and less restraining signing processes. On the other hand in an internet driven world and the rise of crowd sourcing, multi key signature systems can be a first step towards distributed signing. The need for multi key signing in IoT systems incorporating "smart contracts" is also a pressing matter (here identity based signing (Sec.3.5) is heavily used).

References

[1] C. Boyd, "Some application of multi key ciphers," *Advances in Cryptology - EUROCRYPT '88*, 1988.

[2] B. Kaliski, J. Jonsson, and A. Rusch, "Rfc 8017," November 2016, [Online; accessed 07-04-2019].

[3] M. Bellare and G. Neven, "Identity-based multi-signatures from rsa," *Cryptographers' Track at the RSA Conference*, 2007.

[4] A. Bagherzandi and S. Jarecki, "Identity-based aggregate and multi-signature schemes based on rsa," *Public Key Cryptography – PKC 2010*, 2010.

[5] R. Duràn Dìaz and et al., "A review of multisignatures based on rsa," 2010.

[6] D. Boneh and M. Franklin, "Efficient generation of shared rsa keys," *Advances in Cryptology — CRYPTO '97*, pp. 425–439, 1997.

[7] A. Shamir, "How to share a secret," *Communications of the ACM. 22 (11)*, vol. 22, p. 612–613, 11 1979.

[8] G. Blakley, "Safeguarding cryptographic keys," *Managing Requirements Knowledge, International Workshop on (AFIPS)*, 1979.

[9] A. Mauland, "Realizing distributed rsa using secure multiparty computations," July 2009.

[10] S. Tang, "Simple secret sharing and threshold rsa signature schemes," *Journal of Information and Computational Science*, vol. 1, pp. 259–262, 12 2004.

[11] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," *Advances in Cryptology — CRYPTO '91*, pp. 129–140, 1992.