Distributed, Threshold Signing for PKI Root

Lea Kissner
leak@bbn.com

Peiter Zatko
mudge@bbn.com

1 Introduction

In many proposals to attain BGP [6] security [4, 7, 10, 5, 3], such as SIDR [2], an address assignment authority signs resource certificates for address spaces and autonomous system numbers.

Ideally, a single authority would sign every resource certificate. A single signer could easily ensure that the signed certificates did not conflict; for example, no two certificates would assign the same address space to different parties. However, political considerations seem to preclude this solution in the case of address space and autonomous system number resource certificates.

An alternate solution involves each of the five regional internet registries (RIRs) acting as independent certificate authorities. Each RIR would agree to sign certificates only for their own address space. Unfortunately, this approach is far more complex in practice: preventing accidental address space allocation overlap or accidental removal of address allocations across multiple RIRs is hindered in such a stove-piped solution. In addition, at least five public keys (one for each RIR) must be distributed to and updated by all verifiers. While verifiers must obtain and trust some public key in order to verify the validity of certificates, increasing the number of necessary trust anchors greatly increases the complexity of the system for verifiers and can reduce security.

In order to obtain many of the benefits of a single signing authority while allowing the RIRs to retain their autonomy, in this paper we outline procedures to allow multiple players to operate a single *virtual* signing authority as a *root collective*. This virtual signing authority would have a single public key, making it appear as a single signing authority to verifiers, but signing would be cooperatively performed by some subset of the players. The RIRs appear to be operationally suitable to form the root collective. We illustrate the process of creating a signature in Figure 1.

The virtual signing authority must be:

- **Secure.** If the private (signing) key is leaked or otherwise obtained by an adversary, that adversary can sign erroneous data. We refer to such keys as *compromised*. The virtual signing authority must be at least as secure as a single signing authority, including the ability to utilize existing hardware security modules for protection of the secret key. In fact, our scheme for a virtual signing authority can be considered more secure, as more than one secret key must be leaked in order to break the scheme.

- **Invisible.** No verifier should need to know the administrative arrangements of the certificate signing system. Signatures on certificates are verified from a single trust anchor just as if there was a single signing authority instead of a virtual signing authority.

- **Efficient.** Signing certificates should not be a heavy burden on the members of the root collective. In particular, the amount of interactive communication during signing should be minimal: one

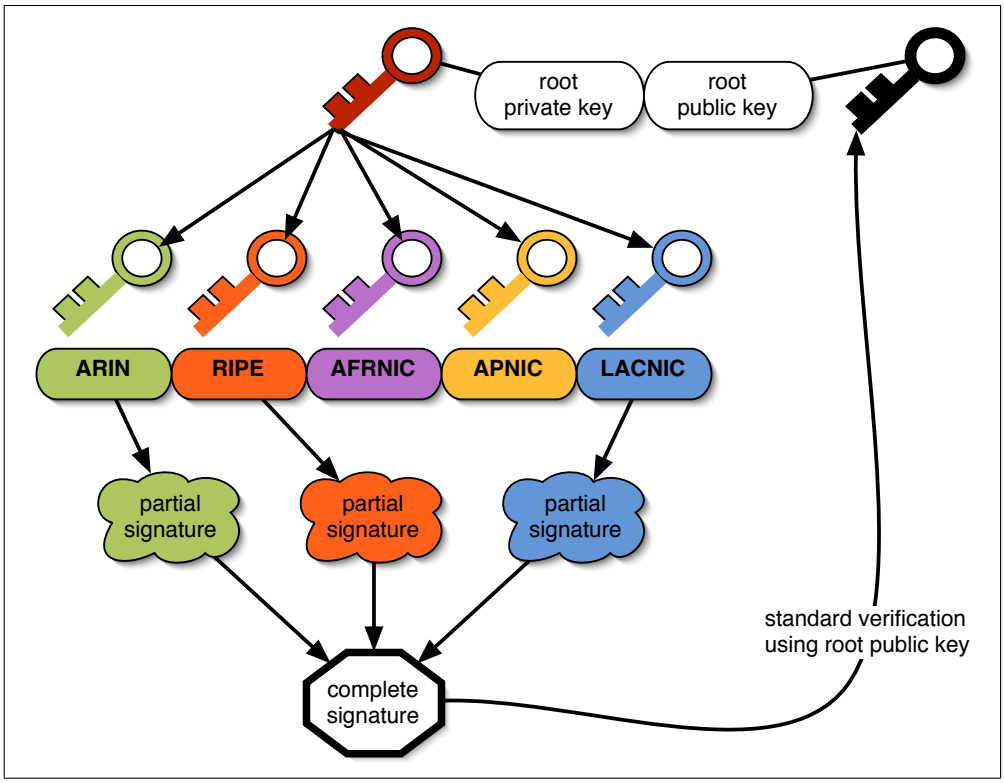

Figure 1: The distributed root of a PKI functions as follows: (1) a trusted dealer chooses a root public and private (signing) key; (2) the dealer creates shares of the signing key for each player/RIR; (3) to create a signature, at least $k = 3$ players create signature shares from their key shares and combine them into a complete signature; (4) the complete signature can be verified with the root public key, just as if it were created directly from the root private key.

player distributes data to be signed and the other players return *signature shares*, which can be combined into a complete signature, valid under the virtual signer's public key. In addition, our scheme for a virtual signing authority can actually increase availability over a single signer, because not all players in the root collective must be available to produce a signature.

2 Concept and Operation of a Distributed Root

The root of a PKI signs certificates for its subordinates; in turn, these subordinates may sign certificates for subordinates of their own[1]. The root public key acts as the ultimate trust anchor within the proposed PKI. As such, it is an extremely tempting target for attack. In a scenario where multiple parties represent the public key, the security of the root private key can be increased by employing a threshold signature scheme to split the root private key among several players. Quorums of root collective members distributed over the globe collectively act as the root signing entity by jointly signing data to be verified

[1]Subordinates may issues certificates only if they are permitted to do so by being specified as a certificate authority in their X509 certificate.

with a single root verification key. Thus the RIRs are responsible for managing address allocations and certificates without requiring IANA to engage in such day-to-day operational issues.

All ℓ participants in the root collective jointly generate key shares for each player such that $k \leq \ell$ players must collaborate to sign data; no group of fewer than k players can sign data. (See Section 3 for details.) At any later time they can sign data as follows: any k players individually construct shares of the signature using their key share; any computer may then combine these signature shares into a valid signature under the root key.

Splitting the root signing key in this way protects against up to $k-1$ players being compromised or attempting to cheat. In combination with strong physical security, the root collective can achieve robust protection of the root signing key while simultaneously distributing control among the participants. To protect against undetected compromise of root collective participants, we recommend they refresh their key shares as we discuss in Section 7. Use of threshold signature can also increase availability, as $\ell - k$ players may be unavailable or slow in response without hindering the signature process.

This paper is a companion to software (see Appendix D) that performs key generation, signing, and verification according to Victor Shoup's threshold RSA signature scheme [9]. In addition, we have added the capability to perform key refresh as we outline in Appendix B.2. Operated according to the procedures described in this document, this code performs the core tasks necessary for a collective of players to operate a virtual signing authority.

3 Overview and Procedures

In order to operate a distributed signing root based on threshold signatures, the ℓ players in the root collective must first determine k, the number of players required to create a signature. This must be chosen such that the maximum number of malicious or compromised players always be less then k. (For our proposal, we recommend $\ell = 5$, $k = 3$.) They then gather to create a common public key; each player gets one share of the secret key. In Section 4, we describe this process, as well as the physical and computational security precautions to be taken.

Each player returns to their operations center with a share of the root signing key on removable media, which they load into the computational device that will construct signature shares. We discuss some of the issues involved in utilizing secured cryptographic signing boxes for this purpose in Section 5.

Any k players of the root collective can now collaborate to sign data such that it can be verified with the root public key. When one player decides to sign some data, that player sends it to all other players. Each of these players examines the data for correct structure and content (e.g., that the address space being allocated is being correctly assigned). Each signing player then uses their secret share of the root signing key to produce a signature share, then sends this share to the requesting player. Once the requesting player has gathered valid signature shares from k distinct players, the requesting player combines them into a signature valid under the root verification key. We show how these signature shares are constructed and combined in Section 6, as well as discuss procedures for producing signatures and identifying malicious or compromised players.

Private key shares should be well protected. However, through administrative error or other means it is possible that a private key share might become known to a malicious entity. To protect against such an entity from obtaining enough private key shares that they can sign data as the root of the PKI, we recommend that the root collective periodically refresh their private key shares. A refresh of private key shares produces new key shares for each one of the participants without requiring a new public key to be issued. This removes the overhead and complexity involved in introducing a new root key unrelated

to previously-issued certificates while maintaining security in regards to known and unknown key share compromise. We discuss several approaches to refreshing key shares in Section 7.

4 How to Generate Keys

The first step in utilizing threshold signatures is to generate the root public key and corresponding private key shares. To ensure the security and integrity of these key shares, all players of the root collective will physically gather for the key generation ceremony. We propose the security guidelines outlined in this section to prevent the key generation process from leaking private information or being biased in favour of any player.

4.1 Key Generation

Representatives of the RIRs meet in person several times per year. These conferences afford the opportunity for 'out-of-band' creation of the root public key and private key shares. Because the RIR representatives have previously met face-to-face, they can avoid the complex issues of remote identity verification by performing key generation during one of these gatherings. In addition, regular physical meetings provide an attractive setting for periodic key refresh, as outlined in Section 7.

To encourage equality and fair play, the root collective first must agree upon a trusted ceremony official. The trusted official will operate the software and hardware involved in the key generation ceremony. Prior to the key generation ceremony, each RIR also creates a public/private RSA key pair exclusively for protecting their private key share during the ceremony.

During the ceremony, the trusted official presents the hardware and software to the RIR representatives for inspection. The official then generates the root public key and private key shares. To give each representative their RIR's key share, the official collects each RIR's public key on removable media, uses each public key to encrypt one key share, and gives each representative the corresponding encrypted key share and root public key on removable media. We recommend the use of encryption to protect the key shares; this prevents participants from obtaining access to other players' key shares. The trusted official then wipes all data from the key generation hardware. We defer details of this procedure to Appendix C.

Before concluding the key generation ceremony, the players may wish to verify that the collective can correctly sign data. The players then protect their key shares through the use of cryptographic and physical security before transporting them to their respective organizations.

In our distributed signature scheme, we utilize the Shoup threshold RSA signature scheme [9]. Key generation for this scheme operates generally as follows:

Input Number of players ℓ, number of players required to create a signature k, key length. (Key length should be chosen to create a secure RSA signature; we recommend at least 4096 bits.)

Output Root public key (e, n), secret key share s_i for each player i $(1 \leq i \leq \ell)$. (In the full scheme, which we do not utilize, players may also receive verification keys $v, v_1, \ldots, v_\ell$. For discussion of proofs of verification for signature shares, see Section 6.)

Any k of the signature shares together define a polynomial that encodes the root secret key; given k of the shares we can reconstruct this secret key (though we do not need to do so to produce signatures) [8].

4.2 Key Transport

Each player must protect the secrecy and integrity of their private key share during transport. It is extremely important that the private key shares not be disclosed either accidentally or to malicious adversaries actively attacking the protection measures. Thus, we recommend that the key shares remain encrypted (and separated from the decryption key) during transit. In addition, the key share should not reside on systems connected to public networks. Participants may wish to incorporate physical security measures, including hardware cryptographic devices and physical locks.

If a player's key is potentially compromised, he must inform the root collective of this fact. If k key shares are compromised, the root key *must* be immediately retired and not used further. If fewer than k keys are compromised, then the root collective can refresh their key shares as soon as possible and make a decision as to how soon they will retire the root key.

By refreshing their key shares, the root collective creates new key shares to all players without changing the root public key. Because of the logistical difficulties entailed in distributing a root key, the root collective should perform key refresh regularly to guard against known and unknown key compromises.

5 How to Utilize a Secure Computing Device

For reasons of security and confidence, we recommend that participants employ secure computing devices for the storage and usage of their secret key shares. Once each player has received their secret key share, the player can transfer it to the secure computing device used to create signatures (see Section 6 for more information on creating signatures). Specifically, player i ($1 \leq i \leq \ell$) who received secret key share s_i at the key generation ceremony (see Section 4) will create standard RSA signatures as his signature shares, using a key of $2s_i\ell!$. (For the five RIRs, $\ell! = 120$.)

Interfaces to such secure devices vary. For this reason, we recommend that the devices that are to be used be PKCS #11 compliant. An example of one such device is SafeNet's Luna SA. The player can load his signing key share into such a HSM. (We give specific directions for SafeNet's Luna devices in Appendix E.2.) The player may then create signature shares as standard RSA signatures using the secret key in the HSM.

6 How to Sign Data

After the players have generated keys as described in Section 4, any k players can jointly sign data.

Creating signature shares. If one player wishes to sign some data, such as a certificate, he sends it to all other players and requests that they create signature shares for this data. (Note that the requesting player needs to receive at least $k - 1$ correct responses.) Those players then check the data according to their established security routines[2].

If a player wishes to help create a signature on the data, that player constructs a signature share. In our code, we utilize Shoup's threshold RSA scheme, so player i ($1 \leq i \leq \ell$) with secret key s_i signs data

[2]For example, the players might institute a rule that they will only sign X.509 certificates that delegate previously unallocated chunks of address space to one RIR, or a rule that they will only sign certificates vouched for by IANA.

x to produce signature share x_i as follows:

$$\begin{aligned} x_n &= x \mod n \\ x_i &= x_n^{2s_i\ell!} \end{aligned}$$

Note that this is a standard RSA signature calculation with secret key $2s_i\ell!$. If the player is utilizing a cryptographic signature box as described in Section 5, then he simply requests that the box produce a signature on data x with key $2s_i\ell!$ to produce the signature share x_i.

Finding corrupted players. The root collective can determine which player(s) are corrupted or malicious by combining signature shares as we discuss in this section; if some players' shares combine into a valid signature, the players who created those shares created them correctly.

If all five participants respond then there are ten possible combinations of the private key shares. If there is exactly one corrupt player then one can create four valid combinations and six invalid combinations. Determining the corrupt player involves identifying the member absent from the valid combinations and verifying that the suspect player player is common within all invalid combinations. In the case of two corrupt players, where all participants have responded, there will be only one valid combination and nine invalid combinations. The players not involved in the valid combination are identifiable as the corrupted players.

We encourage all players to participate and contribute their signature shares even if k participants have already responded. If all players participate, it becomes possible to identify problems or corrupt key shares at an earlier time.

In the full Shoup scheme [9], each player generates a small non-interactive zero-knowledge proof of correctness along with their signature share. These shares do not divulge secret information; they only allow any player to efficiently check that signature shares were constructed correctly. However, in this application, these proofs are not necessary and interfere with the use of hardware security modules. We discuss the security of Shoup's signature scheme without proofs of correctness in Appendix B.1.

Combining signature shares. Once k players have produced signature shares, each player sends their share to the requesting player. Once he has received these shares, the requesting player combines the shares into a joint signature and then checks that the signature is valid under the root public key. If it is not valid, then one of the signature shares is invalid; the player that produced it may be corrupt, incorrect, or an adversary may have interfered with the transmission. The combining player should combine other sets of signature shares, as we describe above, to discover which player (or players) sent an invalid share. Generally, the combination procedure works as follows:

Input Root public key (e, n), signature shares $x_{i_1}, \ldots, x_{i_k}$ from distinct players $i_1, \ldots, i_k$, respectively, data to be signed x.

Output Signature of x under root public key (e, n).

Essentially, the exponents of the signature shares are points from a polynomial that encodes the root secret key. Using polynomial interpolation, we can combine these shares in such a way that the exponent becomes the signing key. In this way, we produce a signature without revealing the secret key. Cryptographic details and mathematical intuition are discussed in Shoup's paper [9].

7 Issues Surrounding Key Refresh

Should it become known that one or more of the key shares (but less than the threshold required to perform signing) have become compromised, key refresh should be initiated. One must also take into account the fact that silent key share compromise might have occurred creating an unknown or undisclosed compromise scenario. To help prevent against both known and unknown compromise we recommend that the root collective refresh their key shares periodically in addition to mandatory refresh events due to known compromise.

One possible key refresh schedule is three to four times per year at the regular meetings of the RIR representatives. Key refresh creates new signing key shares for each player that create signatures valid under the same public key. While they could instead generate shares corresponding a new public key each time, the logistical difficulties involved in distributing the new public key to every single end user makes this approach difficult. By refreshing their key shares, they can protect against an adversary silently gaining access to fewer than k shares of the secret key; the adversary must collect last least k key shares that were created together to forge signatures. Note that, because the public key does not change, attacks based only on knowledge of the public key, such as brute force and factoring, can proceed independent of key refreshing. The root collective must thus change their public key periodically as well, but at much greater intervals.

First, the root collective generates new shares of the secret signing key. Then, they switch to signing with the new shares at an agreed-upon time. It is imperative that each participant destroy their old key shares as soon as possible, as their continued existence creates extra risk. Should an attacker gain access to three key shares created at the same time, the attacker could successfully sign under the existing public key.

The players physically gather and bring their secret keys to the key refresh meeting. Transporting these keys requires physical and computational security measures similar to the key-transport protocols we describe in Section 4.

The trusted official sends out a public key to all players; only the official knows the corresponding secret key. Each player encrypts their key share s_i using this public key and loads it onto removable media, along with the public key required by the share generation procedure.

They then run key refresh software in the same manner as key generation software was run in Section 4. The trusted official loads the encrypted key shares into the computer to be utilized for the key refresh procedure and decrypts them. The official then utilizes the key refresh code to ensure that every key share is correct: if k key shares combine to create a valid private key corresponding to the root public key, those key shares are correct. The trusted official then generates the new key shares using the key refresh code, destroys the old key shares, and distributes the new key shares as in Section 4. We discuss a method for key refresh in Appendix B.2.

8 Conclusion

To increase both availability and security of a PKI root, a collection of responsible parties may share the root signing key between them. They must carefully and jointly generate the key in order to ensure that each gains a secret share of the signing key. When data must be signed, the requesting player sends it to all other players. Those players then decide whether it is valid data that should be signed. If it is, the players each create a signature share from that data and their secret key share and return the signature share to the requesting player. The requesting player then combines the signature shares into a complete RSA

signature valid under the root signing key. In order to guard against the known or unknown compromise of signature shares, the root collective should refresh their key shares periodically. Key refresh allows the players to obtain new key shares without changing the public key. Given the logistical and security issues surrounding public key distribution, key refresh allows more practical recovery compromise of a small number of key shares.

Acknowledgments: We would like to thank Steve Kent for his insightful comments as well as Derrick Kong and Charlie Gardiner for their help in software testing.

References

[1] Java threshold signature package project. `https://sourceforge.net/projects/threshsig`.

[2] Russ Barnes and Stephen Kent. An infrastructure to support secure internet routing.

[3] Y. Hu, A. Perrig, and M. Sirbu. Spv: Secure path vector routing for securing bgp. In *Proceedings of ACM Sigcomm*, August 2004.

[4] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (sbgp). *IEEE Journal on Selected Areas in Communications*, 18(4):58–592, April 2000.

[5] E. Kranakis, P.C. van Oorschot, and T. Wan. On interdomain routing security and pretty secure bgp (psbgp). Technical Report TR-05-08, Carleton University, 2005. `http://www.scs.carleton.ca/research/tech_reports/2005/download/TR-05-08.pdf`.

[6] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). RFC 4271, 2006.

[7] K. Seo, C. Lynn, and S. Kent. Public-key infrastructure for the secure border gateway protocol (s-bgp). In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX)*, June 2001.

[8] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[9] Victor Shoup. Practical threshold signatures. In *Proceedings of Eurocrypt*, 2000.

[10] R. White. Architecture and deployment considerations for secure origin bgp (sobgp). draft-white-sobgp-architecture-01, May 2005.

A Notation

- $a \leftarrow D$ - element a is sampled independently from set D or uniformly at random from set D
- ℓ - number of players that jointly make up the distributed root
- k - number of distributed root players that must collaborate to construct a valid signature
- m - secret for RSA signature scheme; $m = p'q'$
- p - large prime; pq is RSA signature modulus
- q - large prime; pq is RSA signature modulus
- s_i - secret key share for player i $(1 \leq i \leq \ell)$

B Proofs and Mathematical Details

B.1 Regarding Signature Share NIZK Proofs

In Shoup's original scheme [9], proofs of signature share correctness are utilized to determine cheating members of the root collective. We sketch an argument that Shoup's threshold RSA signature scheme [9] remains secure, even in the absence of these proofs of correctness, insofar as a corrupted or malicious player cannot forge a signature.

First, note that by stripping the proofs of correct construction from the signature shares, we are not adding new information or changing the way in which information is conveyed. The behavior of the honest players remains the same; they construct signature shares based on the data to be signed.

Because the behavior of honest players is the same as in Shoup's scheme, we can focus on the ability of the malicious player to construct a signature based on $k-1$ signature shares. As the original scheme is secure and the malicious player receives strictly less information in our scenario, we know that the adversary cannot extract enough information from honest players' signature shares to construct a kth signature share on his own. Because the signature shares are RSA signatures, the shares are existentially unforgeable and the adversary cannot create a valid threshold signature.

B.2 Regarding Key Refresh

B.2.1 Key Refresh Algorithm

Our code utilizes Shoup's threshold RSA signature scheme [9]. Thus, each player i has a secret key share s_i defined as follows according to a polynomial secret sharing scheme, where d, m is the root secret key and $\leftarrow$ indicates a uniform random choice of an element from a set:

$$\begin{aligned} a_0 &= d \\ a_j &\leftarrow \{0, \ldots, m-1\} \\ f(x) &= \sum_{j=0}^{k-1} a_j x^j \\ s_i &= f(i) \end{aligned}$$

Note that we can define this related secret sharing scheme:

$$\begin{aligned} a_0 &= d \\ a_j &\leftarrow \{0, \ldots, m-1\} \\ b_0 &= 0 \\ b_j &\leftarrow \{0, \ldots, m-1\} \\ f_r(x) &= \sum_{j=0}^{k-1} (a_j + b_j) x^j \\ s_{ri} &= f_r(i) \mod m \end{aligned}$$

This produces key shares for each player corresponding to the same secret key as the original scheme, so the players can sign data under the same public root key using shares from the old scheme or the new

scheme. (They cannot, however, mix shares created from different polynomials.) We now construct a key refresh scheme based on this fact:

$$\begin{aligned} b_0 &= 0 \\ b_j &\leftarrow \{0, \ldots, m-1\} \\ f_b(x) &= \sum_{j=0}^{k-1} b_j x^j \\ s_{bi} &= f_b(i) \mod m \\ s'_i &= s_i + s_{bi} \mod m \\ &= \sum_{j=0}^{k-1} (a_j + b_j) x^j \end{aligned}$$

Given k key shares, computer code run by the players can run polynomial interpolation code to reconstruct m. The players thus construct a new random polynomial f_b such that $f_b(0) = 0$, giving each player i ($1 \leq i \leq \ell$) a new key share s'_i. The players begin to use their new key shares to sign data at the time decided upon by the root collective. In this way, the players can refresh their secret key shares without changing the public key.

B.2.2 Security

Next, we prove that our scheme for key refresh is secure.

Our key refresh scheme of Section 7 essentially requires that the players construct polynomial shares of 0 and add them to their original secret shares of the secret key to generate new shares of the secret key. The security of this scheme for uncompromised players follows directly from the security of the polynomial secret sharing scheme [8]. If a player's original secret share remains secret, adding another secret share to the original share does not let the adversary reconstruct it.

We must also ensure that t or fewer players whose shares, $s_{i_1}, \ldots, s_{i_t}$, are known to the adversary are also protected by this scheme. Because some players that were previously corrupted may remain corrupted and some new players may become corrupted during the key refresh process, we allow the adversary to learn at most t shares of the refresh key, $s'_{j_1}, \ldots, s'_{j_t}$.

We must now show by reduction that a player i ($i \in \{i_1, \ldots, i_t\}, i \notin \{j_1, \ldots, j_t\}$, whose secret share is known to the adversary but whose refresh share is not, gains a secret key share through the key refresh process. Note that if the adversary could reconstruct $s_i + s'_i$ with non-negligible probability, the adversary could also reconstruct s'_i with non-negligible probability, as he knows s_i. Given only t or fewer shares of the refresh key and t or fewer shares of the original key, the adversary cannot reconstruct s'_i with non-negligible probability. Thus, the adversary cannot reconstruct player is new key share $s_i + s'_i$ with non-negligible probability.

C Detailed Key Generation Procedures

In this section, we detail a list of requirements and procedures for the trusted ceremony official and the participants.

The trusted official will be provided with the software required to load onto a laptop and perform the key generation in advance of the meeting. Each member will be given copies of the software in advance

and will be permitted to verify that the software that the trusted official will load is the same as the software that they were provided for review.

The trusted official provides:

- A generic laptop computer with CD-ROM drive and USB connection[3], but without a hard disk.
- Software CD-ROM with the requisite OS loader and software that the trusted official will use and that can be examined by the participants prior to start.
- (optional) Hardware key generation device, such as SafeNet Luna PCMCIA.

Each representative from the root collective provides the following:

- A suitably equipped laptop with a CD-ROM drive
- A USB flash drive
- A RSA public key on either a CD-ROM or USB flash drive (this key pair must be generated only for this ceremony; the corresponding private key must be secret)

Once the official and RIR representatives verify that all participants are present and have brought the appropriate resources, setup may commence. All hardware should not be connected to any physical network, have all wireless capabilities turned off or removed, and should remain under the owners' control at all times.

Each representative from the root collective boots their system. They are then provided with the CD-ROM containing the software that the trusted official will be using[4]. The participants are permitted to copy the CD-ROM, examine its contents, and perform any checksum operations they feel necessary to foster a strong belief that the contents of the CD-ROM are correct and consistent with what they had previously been provided.

The official boots their system from the software CD-ROM, which contains a bootable linux kernel that runs in RAM only. Additionally, the CD-ROM contains the required software for key generation, key share splitting, handling the key shares, and sanitization. Sanitization software clears system memory and prevents aversaries from utilizing forensic capabilities to retrieve any portions of the private key shares.

The official loads each of the public keys presented by the members of the root collective onto the system. After this is accomplished the trusted official generates a new RSA public/private key pair and subsequently, using a software application, splits the RSA private key into key shares. The joint secret key is then destroyed. The key shares, one for each member of the root collective, are encrypted using the corresponding player's public key. The resulting encrypted key shares and root public key are loaded onto their respective USB memory sticks and returned to their owners. After the keys have been transferred, the sanitization software is executed to remove any remaining traces of the private key or private key shares from the official's system and the system is powered down.

[3]USB is presented as an example option for removable media. If there are security concerns regarding USB devices, the participants could elect to use one-time recordable CD-ROMs or any other mutually agreed upon media.

[4]The CD-ROM should be **read-only**, alternatively the trusted official can make multiple copies of the CD-ROM in front of everyone (6 in this example), provide a CD-ROM to each participant and retain the final copy for use.

D Code Summary

In this section, we give an overview of Java code [1] for a distributed root based on Shoup's threshold RSA signature scheme [9].

D.1 Key Generation

The root collective players run the following code on a secure computer to generate key shares for each player.

1. Create a new `Dealer` object to create keys of size `keysize`.

   `d = new Dealer(keysize)`

2. Use the Dealer object to generate key shares for ℓ players such that at least k players must collaborate to successfully sign data under the root key.

   `d.generateKeys(`k`, `ℓ`)`

3. Retrieve the root verification key from the `Dealer` object. This key can be utilized to verify all valid signatures constructed by the root collective; it is utilized just as any other RSA public key.

   `gk = d.getGroupKey()`

4. Retrieve the key shares from the `Dealer` object. Each key should be written to a separate piece of media, which is given to the corresponding player.

   `KeyShare[] keys = d.getShares()`

D.2 Constructing Signature Shares

Given their key share `ei` and root key modulus `n`, each player can calculate their signature share as in a standard RSA signature:

```
BigInteger bmod = new BigInteger(b).mod(n)
BigInteger sigshare = bmod.modPow(ei, n)
```

Each of the signing players send `sigshare` and their identifier i to the party combining signature shares into a valid root signature.

Note that our code implements the full Shoup scheme, including non-interactive zero-knowledge proofs. If the root collective were utilizing the full Shoup scheme [9], then each signing player would utilize his KeyShare object `ks` to create a signature share (with associated proof of correctness) on the data `b` as follows: `ks.sign(b)`.

D.3 Combining Signature Shares

Given the data to be signed `b`, an array of k valid signature shares `sigs`, the number of signers k and total number of players ℓ, as well as the root public key object `gk`, one combines the signature shares into a root signature on `b` as follows: `SigShare.combine(b,sigsq,k,l,gk.getModulus(),gk.getExponent())`.

If the root collective were utilizing the full version of Shoup's threshold signature scheme [9], they would have to verify the validity of each share before combining them into a full root signature. Given the data to be signed `b`, an array of k signature shares `sigs`, the number of signers k and total number of players ℓ, as well as the root public key object `gk`, one verifies the validity of the signatures as follows: `SigShare.verify(b, sigs, k, l, gk.getModulus(), gk.getExponent())`.

D.4 Verifying Joint Signature

Given the data that was to be signed `b`, the signature to be checked `sig`, and the root public key `e,n`, one may verify the validity of the signature as with any RSA signature: `sig.modPow(e,n).compareTo(b) == 0`.

E API and Implementation Details

E.1 Hardware Key Generation

While hardware key generation entails greater logistical and security complexity, they can generate cryptographic keys using better entropy sources than available on most computers. The following directions were provided by SafeNet for SafeNet's Luna line of hardware security modules[5]:

1. Generate the key using the PKCS call `C_GenerateKeyPair(... CKM_RSAX_9_31_KKEY_PAIR_GEN, ...`, as this mechanism is FIPS approved

2. Extract the private key from the hardware device.
   `C_GenerateKey( ... CKM_AES_KEY_GEN, ... pTemplate)` (This should set the `CKA_WRAP` and `CKA_DECRPYT` attributes to true and returns the handle of the AES key used to wrap the RSA key.)
   `C_WrapKey(... CKM_AES_CBC_PAD, wrapping key handle, handle of key to wrap)` (This returns the wrapped key blob.)
   `C_DecrypInit(... CKM_AES_CBC_PAD, wrapping key handle)`
   `C_Decrypt(... wrappedKeyBlob, ... , pData, ...)` (`pData` will not contain the decrypted key in DER-encoded form.)

Note that in order to export a key, the key export bit must be set on the hardware generation device. Once set, this bit cannot be cleared without wiping the device. Thus, we recommend using a separate device than the ones used for signing by the root collective.

E.2

Players can import a secret key into SafeNet's Luna devices as follows[6]:

1. Generate a 3DES or AES wrapping key on the HSM with both it's `CKA_ENCRYPT` and `CKA_UNWRAP` bits set

2. DER encode (PKCS#8) the RSA key and encrypt it (`C_Encrypt`) on the HSM with the wrapping key; this call returns the encrypted blob

3. unwrap via `C_Unwrap` the encrypted blob on the HSM using the same wrapping key; this call returns the key handle of the imported RSA key

The public key may be imported into the HSM using the `C_CreateObject` call from PKCS#11.

From this point, the player may create signature shares as standard RSA signatures using the secret key in the HSM.

[5]This information is courtesy of Robert Woodward, Alan Boyd, Mark Yakabuski, and Ben Hanrahan at SafeNet, Inc.
[6]This information is courtesy of Robert Woodward, Alan Boyd, Mark Yakabuski, and Ben Hanrahan at SafeNet, Inc.