Weak Forward Security in Mediated RSA*

Gene Tsudik[1]

Department of Information and Computer Science, University of California, Irvine.
Email: gts@ics.uci.edu

Abstract. Mediated RSA (mRSA) [1] is a simple and practical method of splitting RSA private keys between the user and the Security Mediator (SEM). Neither the user nor the SEM can cheat each other since a signature or a decryption must involve both parties. mRSA allows fast and fine-grained control (revocation) of users' security priviliges.
Forward security is an important and desirable feature for signature schemes. Despite some notable recent results, no forward-secure RSA variant has been developed. In this paper (abstract), we show how weak forward security can be efficiently obtained with mediated RSA. We consider several methods, based on both multiplicative and additive mRSA and discuss their respective merits.

1 Forward Security

Forward security is a timely and active research topic which has received some attention in the recent research literature. The purpose of forward security is to mitigate an important problem in ordinary public key signatures: the inability to preserve the validity of past signatures following a compromise of one's private key. In other words, if a forward-secure signature scheme is employed, an adversary who discovers the private key of a user is unable to forge user's signatures from earlier times (pre-dating the compromise).

The notion of forward security was introduced by Anderson [2]. Since then, a number of schemes were proposed. Some are generic, i.e., applicable to any signature scheme [3], while others target (and modify) a particular signature scheme to achieve forward security [4, 5].

In this paper we concentrate on weak forward security in a mediated signature setting. Informally, weak forward security means that the adversary is unable to forge past signatures if she compromises only one (of the two) share-holders of the private key. Specifically, we propose, discuss and analyze two simple schemes built on top of mediated RSA (mRSA), a 2-out-of-2 threshold RSA scheme.

The paper is organized as follows: the next section provides an overview of mRSA. Then, Section 3 describes the forward secure additive mRSA and discusses its security and efficiency features. Section 4 presents another scheme based on multiplicative mRSA. This scheme is more flexible but slightly less efficient. The paper ends with the summary and some directions for future work.

* This work was supported by DARPA contract F30602-99-1-0530.

2 Mediated RSA

Mediated RSA (mRSA) involves a special entity, called a SEM (SEcurity Mediator) which is an on-line semi-trusted server. To sign or decrypt a message, Alice must first obtain a message-specific token from the SEM. Without this token Alice can not use her private key. To revoke Alice's ability to sign or decrypt, the administrator instructs the SEM to stop issuing tokens for Alice's public key. At that instant, Alice's signature and/or decryption capabilities are revoked. For scalability reasons, a single SEM serves many users. One of the mRSA's advantages is its transparency: SEM's presence is invisible to other users: in signature mode, mRSA yields standard RSA signatures, while in decryption mode, mRSA accepts plain RSA-encrypted messages.

The main idea behind mRSA is the splitting of an RSA private key into two parts as in threshold RSA [6]. One part is given to a user while the other is given to a SEM. If the user and the SEM cooperate, they employ their respective half-keys in a way that is functionally equivalent to (and indistinguishable from) standard RSA. The fact that the private key is not held in its entirety by any one party is transparent to the outside, i.e., to the those who use the corresponding public key. Also, knowledge of a half-key cannot be used to derive the entire private key. Therefore, neither the user nor the SEM can decrypt or sign a message without mutual consent.

We now provide an overview of mRSA functions. The variant described below is the additive mRSA (+mRSA) as presented by Boneh, et al. in [1]. There is also a multiplicative mRSA variant – denoted *mRSA – where the private key is computed as the product of the two shares. (See Appendix for the description). Muliplicative mRSA was first introduced in the Yaksha system [7] and later discussed in [8].

Algorithm +mRSA.key (executed by CA)

Let k (even) be the security parameter

1. Generate random $k/2$-bit primes: p, q
2. $n \leftarrow pq$
3. $e \stackrel{r}{\leftarrow} Z^*_{\phi(n)}$
4. $d \leftarrow 1/e \bmod \phi(n)$
5. $d_u \stackrel{r}{\leftarrow} Z_n - \{0\}$
6. $d_{sem} \leftarrow (d - d_u) \bmod \phi(n)$
7. $SK \leftarrow d$
8. $PK \leftarrow (n, e)$

After computing the above values, the CA securely communicates d_{sem} to the SEM and d_u – to the user. (A detailed description of this procedure can be found in [1].) The user's public key PK is released, as usual, in a public key certificate.

Protocol +mRSA.sign (executed by User and SEM)

1. USER: $h \leftarrow H(m)$
   where $H()$ is a suitable hash function (e.g., SHA-based HMAC) and $|H()| < k$.
2. USER: send h to SEM.
3. In parallel:
   3.1 SEM:
   (a) If USER revoked return (ERROR)
   (b) $PS_{sem} \leftarrow h^{d_{sem}} \bmod n$
   (c) send PS_{sem} to USER
   3.2 USER:
   (a) $PS_u \leftarrow h^{d_u} \bmod n$
4. USER: $h' \leftarrow (PS_{sem} * PS_u)^e \bmod n$
5. USER: If $h' \neq h$ then return (ERROR)
6. $S \leftarrow (PS_{sem} * PS_u) \bmod n$
7. USER: return (h,S)

The signature verification (+mRSA.ver) algorithm is not provided as it is identical to that in plain RSA.

3 Forward Secure +mRSA

The main idea in forward-secure additive mRSA (FS+mRSA) is for both SEM and user to evolve their private key shares in parallel. The evolution is very simple: each party logically multiplies its share by e. We say "logically" since no actual muliplication is performed; instead, each party merely maintains a counter (i) which is the index of the current time period. As in all other forward secure schemes, there is a maximum number T past which the shares are not evolved.

At any given time, the current private key is:

$$d_i = d_0 * e^i \text{ and } d_0 = d * e^{-T}$$

The user's and SEM's respective key shares, at a given interval are:

$$d_{i,u} = (d_{0,u}) * e^i \text{ where } d_{0,u} = d_u * e^{-T} \bmod \phi(n)$$

and:

$$d^{i,sem} = (d_{0,sem}) * e^i \text{ where } d_{0,sem} = d_{sem} * e^{-T} \bmod \phi(n)$$

The i-th private key evolution can be thus rewritten as:

$$d_i = (d_{0,u}) * e^i + (d_{0,sem}) * e^i = d_0 * e^i = d * e^{i-T}$$

In actuality, both user and SEM always maintain $d_{0,u}$ and $d_{0,sem}$, which are their respective initial shares. However, when they apply their respective shares (to sign a message) they use the current evolution. The reason for not actually

computing $d_{i,u/sem}$ is because neither the SEM nor the user knows $\phi(n)$ and thus cannot compute values of the form:

$$((d_{0,u/sem}) * e^i) \mod \phi(n)$$

Recall that p, q and, consequently, $\phi(n)$ are known only to the CA.

The flavor of forward security offered by our approach is *weak*. Here 'weak' means that, throughout the lifetime of the public key (T periods), the adversary is allowed to compromise only one of the parties' secrets, i.e., only $d_{i,u}$ or $d_{i,sem}$ but not both.

Although the above may be viewed as a drawback, we claim that weak forward security is appropriate for the mRSA setting, since the security of mRSA is based on the non-compromise of both key shares. More specifically, the SEM is an entity more physically secure and more trusted than a regular user. Hence, it makes sense to consider what it takes for mRSA to be forward secure primarily with respect to the user's private key share.

3.1 FS+mRSA in detail

Like most forward-secure signature methods, FS+mRSA is composed of the following four algorithms: `FS+mRSA.key, FS+mRSA.sign FS+mRSA.ver and FS+mRSA.update`. The purpose of the first three is obvious, while `FS+mRSA.update` is the secret key share update algorithm executed by both user and SEM. We do not specify `FS+mRSA.update` since it is trivial: as mentioned above, it does not actually evolve each party's key share: it merely increments the interval counter.

`Algorithm FS+mRSA.key (executed by CA)`

`Let` (t, T) `be the length of the update interval and the max. number of update intervals, respectively.`

1-7. Identical to +mRSA.key
8. $PK \leftarrow (t, T, n, e)$
9. $d_{0,u} \leftarrow d_u * e^{-T} \mod \phi(n)$
10. $d_{0,sem} \leftarrow d_{sem} * e^{-T} \mod \phi(n)$

Protocol FS+mRSA.sign (i,m)

i (0 $\leq$ i $\leq$ T) is the current interval index and m is the input message

1. USER: $h \leftarrow H_i(m)$
   where $H_i()$ is a suitable hash function (e.g., SHA-based HMAC) indexed with the current interval. ($|H_i()| < k$)
2. USER: send m to SEM.
3. In parallel:
   3.1. SEM:
   (a) If USER revoked return (ERROR)
   (b) $h \leftarrow H_i(m)$
   (c) $PS_{sem} \leftarrow (h^{d_{0,sem}})^{e^i} \bmod n$
   (d) send PS_{sem} to USER
   3.2. USER:
   (a) $PS_u \leftarrow (h^{d_{0,u}})^{e^i} \bmod n$
4. USER: $h' \leftarrow (PS_{sem} * PS_u)^{e*e^{T-i}} \bmod n$
5. USER: If $h' \neq h$ then return (ERROR)
6. $S \leftarrow (PS_{sem} * PS_u) \bmod n$
7. USER: return (h,S)

We note that, in steps 3.1.a, 3.2.b and 4, two exponentiations are performed.

Algorithm FS+mRSA.ver (i,S,m,e,n)

i ($0 \leq i \leq T$) is the claimed interval index,
S is the purported signature on a message m, and (e, n) is the public key of the signer

1. if ($i < 0$) or ($i > T$) return (ERROR)
2. $h \leftarrow H_i(m)$
3. $h' \leftarrow S^{e*e^{T-i}}$
4. If $h' \neq h$ then return (0)
5. return (1)

From the descriptions of FS+mRSA.sign and FS+mRSA.ver it is clear that the present scheme is correct, i.e., signature verification succeeds iff a valid signature is provided:

$$S \leftarrow h^{d^i} = h^{d_{i,u}+d_{i,sem}} = h^{d_u*e^{-T}*e^i+d_{sem}*e^{-T}*e^i} = h^{d*e^{i-T}}$$

and

$$S^{e*e^{T-i}} = (h^{d*e^{i-T}})^{e*e^{T-i}} = h^{ed} = h$$

3.2 Efficiency

In [1], the efficiency of mRSA is shown to be roughly equivalent to unoptimized RSA, i.e., RSA without using the Chinese Remainder Theorem (CRT). The

efficiency of FS+mRSA is only slightly lower than that of mRSA. The only difference in signing is the extra exponentiation with e^i performed by both the user and the SEM in parallel.

In general, an additional exponentiation with e^i is also needed for verifying FS+mRSA signatures. However, we observe that, if the user's public exponent is small (e.g., 3), the curent public key $e_i = 3^{i+1}$ is likely to be smaller than the modulus n for many values of i. For example, if $e = 3$ and $k = 1024$, $|e_i| < k$ and $e_i < n$ for $0 \leq i \leq 592$. In that case, e_i can be stored as a single $k - bit$ number and only one exponentiation would be required to verify an FS+mRSA signature.

The extra storage due to forward security in FS+mRSA is negligible. Since key shares are only logically evolved, the only extra information maintained by all signers and SEM-s (on top of what is already required by mRSA) is the index of the current time interval.

3.3 Security Considerations

In all security aspects (other than forward security) the proposed scheme is essentially equivalent to plain RSA as argued in [1]. Similarly, the *forward security* property of FS+mRSA is based on the difficulty of computing roots in a composite-order group which is also the foundation of the RSA cryptosystem. While this extended abstract does not contain a proof of this claim, we provide the following informal argument:

> Assume that the adversary compromises the user at an interval j and, as a result, learns $d_{0,u}$.[1] In order to violate forward security, it suffices for the adversary to generate a single new signature of the form (where $i < j$ and $h = H(m)$ for some new message m):
>
> $$S = h^{d_i} = h^{d_0 * e^i} = h^{d_{0,sem} * e^i + d_{0,u} * e^i} = (h^{d_{0,sem} * e^i} * h^{d_{0,u} * e^i})$$
>
> Computing $h^{d_{0,u} * e^i} \pmod n$ is trivial. However, computing $h^{d_{0,sem} * e^i} \pmod n$ seems to require taking e-th (cube) roots (mod$n$) since, in the current interval j ($j > i$), the SEM is using as its key share ($d_{0,sem} * e^j$) and only "produces" values of the form: $h^{d_{0,sem} * e^j} \bmod n$.

All forward-secure signature schemes proposed thus far rely on the secure deletion of old secret keys. This is not always a realistic assumption, especially when secure (tamper-resistant) hardware is not readily avaialable. In this aspect, FS+mRSA offers an advantage since the user's secret key share is not actually evolved and no deletion of prior secrets is assumed. While the compromise of the user's current key share yields all user's key shares for all prior intervals, no past-dated signatures can be produced since the SEM's key share evolves separately. We also note that this property is symmetric: if a SEM's key share

[1] This is possible because $d_{j,u}$ is never actually computed, but composed, when needed, as $d_{0,u} * e^j$.

is ever compromised, forward security is preserved as long as the user's share remains secure.

There are, however, two types of attacks unique to FS+mRSA. We refer to the first type as a *future-dating attack*. In it, an adversary obtains a valid signature from the user (m, S) under the current public key (e, n, i). He then takes advantage of the private key structure to construct a valid signature (S') on the same message m dated in some future interval j $(i < j \leq T)$. This can be easily done by computing:

$$S' = S^{e^{j-i}} = (h^{(d_u + d_{sem}) * e^{i-T}})^{e^{j-i}} = h^{d * e^{j-T}}$$

We note that this attack does not compromise the forward security property of the signature scheme. However, it does pose a problem for FS+mRSA. Fortunately, there is an easy fix. It involves hashing the index of the current time interval together with the message. This is already specified in the initial step in protocol `FS+mRSA.sign`. (In other words, instead of $h = H(m)$ we can compute $h = H(m, i)$, or, better yet, $h = \mathrm{HMAC}_i(m)$. This essentially rules out the future-dating attack.)

The second attack type is an *oracle attack*. In it, an adversary, masquarading as the user, sends signature requests to the SEM during the time interval i. This is easy to do since the communication channel between the user and the SEM is neither private nor authentic. The adversary collects a number of "half-signatures" of the form (m, PS_{sem}) where:

$$PS^{i,sem} = h^{d_{sem} * e^{i-T}}$$

Suppose that at a later interval j $(i < j \leq T)$, the adversary actually compromises the user's secret $d_{j,u}$. Although the adversary can not compute "new" signatures from prior time intervals, he can use the previously acquired half-signatures to forge signatures from period i:

$$S' = (PS_{i,sem})^{d_u * e^{i-T}} = (h^{d_{sem} * e^{i-T}})^{d_{sem} * e^{i-T}} = h^{d * e^{i-T}}$$

One simple way of coping with the oracle attack is to require the user-SEM communication channel to be *authentic*. This is not much of a burden in practice due to widely-deployed and available tools such as IPSec [9] and SSL [10]. An alternative is to require mRSA-based authentication of the signature request messages flowing from the user to the SEM. This can be accomplished as follows. When sending a signature request to the SEM, the user computes, in addition: $\bar{h} = \bar{H}(h)$ where $\bar{H}() \neq H()$ is a suitable (cryptographically strong) hash function such that $|\bar{H}()| < k$. He then computes:

$$P\bar{S}_u \leftarrow \bar{h}^{d_{0,u}} \bmod n$$

The user sends $P\bar{S}_u$ along with h in the signature request to the SEM. The SEM, before computing its half-signature (PS_{sem}), verifies $P\bar{S}_u$ by computing: $\bar{h}' = \bar{H}(h)$ and comparing:

$$\bar{h}' \text{ and } (P\bar{S}_u{}^{d_{0,sem}})^{e * e^T} \bmod n$$

Since these two values match only if the user originated the signature request, oracle attacks can be thereby avoided.

4 Forward Secure *mRSA

We now construct another forward-secure scheme based on the multiplicative mRSA variant. Only the key generation and signing algorithms are shown; the verification algorithm is identical to that in FS+mRSA.

Algorithm FS*mRSA.key

1-7. Identical to *mRSA.key (see Appendix)
8. $PK \leftarrow (t, T, n, e)$
9. $d_{0,sem} \leftarrow d_{sem} * e^{-T} \bmod \phi(n)$

The main difference with respect to $FS + mRSA$ is the unilateral update feature. In the present scheme, only the SEM's share is evolved whereas the user's share remains the same throughout the lifetime of the key. This is a desireable feature since it saves the user one exponentiation over FS+mRSA. However, this does not significantly influence the overall performance since, unlike FS+mRSA, the two parties cannot compute their half-signatures in parallel.

Protocol FS*mRSA.sign

1. USER: $h \leftarrow H_i(m)$
2. USER: send m to SEM
3. SEM: If USER revoked return (ERROR)
4. SEM: $h \leftarrow H_i(m)$
5. SEM: $PS_{sem} \leftarrow h^{d_{i,sem}} \bmod n$
6. SEM: send PS_{sem} to USER
7. USER: $h' \leftarrow (PS_{sem}^{d_u})^{e*e^{T-i}} \bmod n$
8. USER: If $h' \neq h$ then return (ERROR)
9. $S \leftarrow (PS_{sem}^{d_u}) \bmod n$
10. USER: return (h,S)

The correctness of FS*mRSA is evident from the verification procedure:

$$(S)^{e*e^{T-i}} = ((PS_{sem})^{d_u})^{e*e^{T-i}} = ((h^{d_{i,sem}})^{d_u})^{e*e^{T-i}} =$$
$$((h^{d_{sem}*e^{i-T}})^{d_u})^{e*e^{T-i}} = (h^{d*e^{i-T}})^{e*e^{T-i}} = h$$

Just like FS+mRSA, this scheme is vulnerable to both future-dating and oracle attacks. Fortunately, the exact countermeasures described in Section 3.3 are equally applicable here.

Another trivial variation of this scheme entails only the user (but not the SEM) evolving its key share. This is a less attractive option since it is much more likely that the user, rather than the SEM, succumbs to eventual compromise. Finally, it is also possible to have both parties evolving the key (just as in FS+mRSA). The main difference here would be that signature verification would require an extra exponentiation with e^{2i} rather than e^i.

5 A Final Observation

A crucial (but purposely ignored above) detail in the construction of `FS+mRSA` and `FS*mRSA` is the use of the current period index i in the hashing of the input message. The intended purpose of the index is as a hedge against possible attacks against the key evolution scheme. However, a closer look indicates that the inclusion of the index in the hash is **in and of itself** sufficient to provide the same weak forward security that we hope to attain with key evolution. In this case, key evolution as described above can be dropped completely to be replaced by the simple hashing of the period index. (This would also result in a much more efficient scheme.)

6 Summary

We described two methods of obtaining efficient (yet weak) forward security with mediated RSA. These methods work with both multiplicative and additive mRSA variants. The degree of forward security is weak since we assume that only the user or the SEM (but not both) are compromised by the adversary. However, this assumption is in line with the mRSA notion of security which is based on the inability to compromise both parties.

Since, aside from signatures, mRSA can be used for encryption, the natural issue to consider is whether FS+mRSA and FS*mRSA schemes are useful for *forward-secure encryption*.

7 Acknowledgements

Many thanks to Giuseppe Ateniese for pointing out an attack on the previous version of `FS+mRSA` as well to anonymous referees for their comments.

References

1. D. Boneh, X. Ding, G. Tsudik, and B. Wong, "Instanteneous revocation of security capabilities," in *Proceeding of USENIX Security Symposium 2001*, Aug. 2001.
2. R. Anderson, "Invited lecture at the acm conference on computer and communication security (ccs'97)," 1997.
3. H. Krawczyk, "Simple forward-secure signatures from any signature scheme," in *ACM Conference on Computer and Communication Security (CCS'00)*, 2000.
4. G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," in *CRYPTO'01*, 2001.
5. M. Bellare and S. Miner, "A forward-secure digital signature scheme," in *CRYPTO'99*, 1999.
6. P. Gemmel, "An introduction to threshold cryptography," *RSA CryptoBytes*, vol. 2, no. 7, 1997.
7. R. Ganesan, "Augmenting kerberos with pubic-key cryptography," in *Symposium on Network and Distributed Systems Security* (T. Mayfield, ed.), (San Diego, California), Internet Society, Feb. 1995.

8. P. MacKenzie and M. K. Reiter, "Networked cryptographic devices resilient to capture," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pp. 12–25, May 2001.
9. S. Kent and R. Atkinson, "RFC 2401: Security architecture for the internet protocol," Nov 1998.
10. "The openssl project web page," *http://www.openssl.org*.

A Multiplicative mRSA – *mRSA

The *mRSA variant is largely similar to its additive counterpart. The only substantive difference has to do with parallelizing private key operations by the user and the SEM. In +mRSA, both parties can perform exponentiations with their respective private key shares in parallel. In contrast, *mRSA prescribes serial operation.

Algorithm *mRSA.key (executed by CA)

Let k (even) be the security parameter

1. Generate random $k/2$-bit primes: p, q
2. $n \leftarrow pq$
3. $e \stackrel{r}{\leftarrow} Z^*_{\phi(n)}$
4. $d \leftarrow 1/e \bmod \phi(n)$
5. $d_u \stackrel{r}{\leftarrow} Z^*_{\phi(n)}$
6. $d_{sem} \leftarrow (d/d_u) \bmod \phi(n)$
7. $SK \leftarrow (d, n)$
8. $PK \leftarrow (n, e)$

As in additive mRSA, CA securely communicates d_{sem} to the SEM and d_u – to the user. The user's public key PK is released as part of a public key certificate.

Protocol *mRSA.sign (executed by User and SEM)

1. USER: $h \leftarrow H(m)$
2. USER: send h to SEM
3. SEM: If USER revoked return (ERROR)
4. SEM: $PS_{sem} \leftarrow h^{d_{sem}} \bmod n$
5. SEM: send PS_{sem} to USER
6. USER: $h' \leftarrow (PS^{d_u}_{sem})^e \bmod n$
7. USER: If $h' \neq h$ then return (ERROR)
8. USER: $S \leftarrow (PS^{d_u}_{sem}) \bmod n$
9. USER: return (h,S)