Improvements to the Merkle signature scheme

(extended abstract)

Luis Carlos Coronado García
FB 20, Technische Universität Darmstadt
Hochschulstr. 10, D-64289, Darmstadt. Germany.
coronado@cdc.informatik.tu-darmstadt.de
http://www.cdc.informatik.tu-darmstadt.de/mitarbeiter/coronado.html

1 Introduction

We propose two efficient and secure versions of the Merkle signature scheme. One of them is forward secure and its security is based on that of its primitives: a collision resistant and one way hash function and a cryptographic secure pseudorandom bit generator. In the other version; the private key is composed of two separated parts: a user-key and a home-key. These parts of the private key evolve during several periods of the lifetime of the instance of the scheme, while the public key remains the same. The goal of the second version is to prevent forged signatures in all periods except for those where only the user-key is compromised. In those periods where only the home-key is compromised, the intruder cannot forge any signature at all. In case that the home-key and the user-key are compromised, the scheme result in a forward secure one.

These two versions are quite efficient, existentially unforgeable under adaptive chosen message attack and have forward security.

2 Brief description of the Merkle signature scheme

Merkle presents in [Mer90] a multi-time signature scheme. Actually, the Merkle signature scheme (MSS) transforms any one-time signature scheme in a multi-time one. Because of the constraint in the number of pages in the extended abstract we suggest the reader to consult [Mer90] for a detailed description of the original scheme. In the following we provide a brief description of an improved version of the MSS for creating up to 2^N signatures. The ***key generation algorithm*** $MGen(s, N)$ works as follows: On input the security parameter s (in unary) and N, it generates 2^N key pairs of the one-time signature scheme (in case of the use of a cryptographically secure pseudorandom bit generator $prg : \{0,1\}^s \rightarrow \{0,1\}^{2s}$, it can be obtained $(PK_i^{\text{ots}}, PS_i^{\text{ots}}) \leftarrow Gen_{\text{ots}}(s, seed_i); (\chi_i, seed_i) \leftarrow prg(\chi_{i-1})$ for $0 \leq i < 2^N$ and $\chi_{-1} \in_R \{0,1\}^s$). Then, it creates a binary tree, which will be called Merkle tree henceforth, whose leaves are the hash values of the verification keys, and each parent is the hash value of the concatenation of its left and right children. It outputs the public (PK) and private (SK) keys. The public key is the depth and the root of the Merkle tree (N, R). All the one-time key pairs taken together serve as the private key or, in case of the use of prg, only $\chi = \chi_{-1}$ can be the private key. The ***signature algorithm*** $MSig(M, SK)$ of the k-th message is the one-time signature τ (made by the one-time signature algorithm with the k-th one-time private key), followed by the corresponding one-time verification PK_{ots} key and N nodes $P_N, \ldots, P_1$ from the Merkle tree which help to authenticate the verification key against PK. The ***verification algorithm*** $MVer(M, (i, \tau, PK_{\text{ots}}, P_N, \ldots, P_1), PK)$ employs the one-time

verification algorithm with M, τ and PK_{ots}, then it verifies the authenticity of the one-time verification key PK_{ots} using i, the auxiliary nodes $P_N, \ldots, P_1$ and PK.

3 A forward secure and efficient version of the Merkle signature scheme

A sketch of proof of security. Under the assumption of the existence of cryptographically secure hash functions and cryptographically secure one-time signature schemes, the MSS is not existentially forgeable under adaptive chosen message attack. The idea to do this is: prove first that any alteration of any number of nodes of a Merkle tree that keeps the root intact yields an explicit collision for the underlying hash function. Then, show that any existential forgery of signatures in the MSS leads to either an existential forgery of signatures for the underlying one-time signature scheme or a collision for the underlying hash function.

A forward secure version of the Merkle Signature Scheme. Roughly speaking, a forward secure signature scheme is one for which the validity of a public key is divided into periods, and the corresponding private key "evolves" after each period in such a way that if the private key is compromised by an adversary in some period, the adversary cannot succeed in forging a signature for a previous period. We modify the original MSS in order to transform it into a forward secure one. The description of such an improved scheme is given in Section 2. Note that if all the one-time signing keys are stored and each one of them is deleted after its use, the resulting MSS is forward secure, where each period consists of exactly one signature. In this case, the private key of the MSS is as big as the stored one-time signing keys. In our version, the size of the private key is reduced by employing a pseudorandom bit generator and a one-time signature scheme with deterministic key-generation. Bellare and Yee have shown in [BY03] how to construct a forward-secure pseudorandom bit generator from a cryptographically secure pseudorandom bit generator. The use of that bit generator enables us to prove that our new version of the MSS becomes forward secure.

Key generation process split through the signature process. In the MSS, the number of possible signatures is 2^N, where N is the depth of the Merkle tree. The bigger the parameter N is, the slower the key generation process becomes: during key generation $2^{N+1} - 1$ hash values and 2^N one-time key pairs have to be computed.

In our version, part of the key generation takes place during the use of the signature scheme. This permits the use of sufficiently large parameters to allow for a practically unlimited number of signatures while keeping the cost of the initial key generation process low. The basic idea is to use one ("top") Merkle tree to authenticate the roots of a series of other, "bottom" trees. Only one of the bottom trees is kept at a time. During the use of one bottom tree, the next one is generated, namely two nodes at a time per signature.

Let $MSign = (MGen, MSig, MVer)$ be the MSS as described in Section 2. We describe a new and efficient version of the MSS, which has –in the practical sense– an unlimited number of possible signature.

Improved key generation algorithm *Gen*. On input the security parameter s and the parameter N, for 2^{2N} possible signatures, *Gen* calls twice $MGen$ to compute $(\chi_{-1}, (N, R))$ and $(\chi_{0,-1}, (N, R_0))$ and then computes $\zeta_0 = MSig(R_0, \chi_{-1})$ and outputs (N, R) as the public key and $(\chi_{0,-1}, \chi_0)$ as the private key, where χ_0 is obtained from $(\chi_0, seed_0) \leftarrow prg(\chi_{-1})$. The signer must keep two counters. One of them (i) counts the number of generated signatures modulo 2^N, which at this point must be initialized to zero, and the other (j) counts the number of signatures created by the first generated Merkle key pair, which at this point must be initialized to zero, too. The signer must also keep (ζ_0, R_0), which is the signature of the public key of the second

generated key pair and the public key itself.
Improved signature algorithm Sig**.** Let i and j be the counters described in the previous improved key generation algorithm. On input the secret key $(\chi_{j,i-1}, \chi_j)$ and a message M, Sig computes $\tau_i \leftarrow MSig(M, \chi_{j,i-1})$ and sets $\sigma = (\tau_i, R_j, \zeta_j)$, then obtains $\chi_{j,i}$ from $(\chi_{j,i}, seed_{j,i}) \leftarrow prg(\chi_{j,i-1})$ and, after that, increments i by one. If at this point $i \cong 0 \mod 2^N$, Sig calls $MGen$ to obtain $(\chi_{j+1,-1}, (N, R_{j+1}))$ then computes $\zeta_{j+1} \leftarrow MSig(R_{j+1}, \chi_j)$ and χ_{j+1} from $(\chi_{j+1}, seed_{j+1}) \leftarrow prg(\chi_j)$. Sig sets $i \leftarrow 0$ and increments j by one. Finally, Sig outputs the signature σ. The signer must keep (ζ_j, R_j), too.
Improved verification algorithm Ver**.** On input a signature $\sigma = (\tau, \rho, \zeta)$ and a message M, Ver accepts the signature if both $MVer(M, \tau, (N, \rho))$ and $MVer(\rho, \zeta, (N, R))$ are $true$, and rejects it otherwise.

4 Experimental Results

Our experiments are estimates of the size of the Merkle keys and MSS and also estimates of the time needed for the Key Generation, Signature and Verification algorithms. These experiments were made on a SUN 4 ultra SPARC Sun-Blade-100, Sun OS 5.8, at 500MHz and we have used RIPEMD160 as hash function, our improved version of the Lamport-Diffie one-time signature scheme, which is described in Appendix A, and the number of possible signatures is 2^N. The cryptographic library used in the computations of RIPEMD160 values is OpenSSL [Ope] version 0.9.7d. and the used pseudorandom bit generator is ISAAC [ISA].

RSA Signature Scheme		
key size (bits)	Signature	Verification
	milliseconds	
512	28.01	1.08
1024	55.52	2.09
2048	224.55	6.50

	Key Generation		Signature		Verification
N	Time	size Kb	Time	size Kb	Time ms
16	3.46 sec	1.57	446 ms	3.93	2.96
18	6.9 sec	2.31	616 ms	3.97	3.12
20	13.8 sec	2.43	682 ms	4.01	3.27
22	27.6 sec	3.8	1.04 sec	4.05	3.43
24	55.2 sec	3.91	1.13 sec	4.09	3.59
26	1.84 min	6.53	1.9 sec	4.13	3.74
28	3.68 min	6.65	2.05 sec	4.17	3.9
30	7.36 min	11.8	3.77 sec	4.21	4.05
32	14.7 min	11.9	4.02 sec	4.25	4.21
34	29.4 min	22	7.85 sec	4.29	4.37
36	58.9 min	22.1	8.31 sec	4.32	4.52
38	1.96 hrs	42.2	16.8 sec	4.36	4.68
40	3.92 hrs	42.4	17.6 sec	4.4	4.83

Table 1: Timing for RSA and for our first improvement to the Merkle signature scheme: Key Generation Time, Private Key Size, Signature Process Time, Signature Size and Verification Time. The number of possible signatures is 2^N and the public-key size is 24 bytes.

5 Second version of the Merkle signature scheme

Suppose that $H : \{0,1\}^* \to \{0,1\}^s$ is a hash function and $prg : \{0,1\}^s \to \{0,1\}^{2s}$ is a pseudo-random bit generator. In the following we roughly sketch our scheme:
Key Generation Algorithm. There exist two entities: The home-entity and the user-entity. The user-entity computes an instance of the multi-time signature scheme (sk^{mt}, pk^{mt}), selects

a random $r \in \{0,1\}^s$ and sends (r, pk^{mt}) to the home-entity. The home-entity computes an instance of the MSS (SK_M, PK_M), obtains τ (the signature of pk^{mt} with SK_M), selects a random $r' \in \{0,1\}^s$ and finally computes $sk_M \leftarrow SK_M \odot (r \oplus r')$. $x \oplus y$ is the xor'ing of x and y als bit strings of the same length and $SK_M \odot \rho$ is the xor'ing of α_n with $H^n(\rho)$; here $SK_M = \|_{n \in I} \alpha_n$, each α_n has length s and $\|$ is the concatenation of strings. The home-entity returns (r', τ) to the user-entity and deletes r and r'. The user-entity sets $r \leftarrow r \oplus r'$ and stores $(r, \tau, pk^{mt}, sk^{mt})$. The home-entity stores sk_M. The public key is PK_M.

Signature Algorithm. On input a message m, the user-entity computes τ_u, the signature of the message m with the private key sk^{mt}. Then outputs $\sigma = (\tau_u, \tau, pk^{mt})$. The current period is explicit in the Merkle signature τ.

Verification Algorithm. On input a message m, a signature $\sigma = (\tau_1, \tau_2, pk)$ and the public key PK_M, the verifier employs the multi-time verification algorithm for m and τ_1 and then the Merkle verification algorithm for τ_2 and pk $(MVer(pk, \tau_2, PK_M))$. The verifier accepts the signature if both outcomes are true or rejects it in other case.

Key Update Algorithm. The user-entity deletes (pk^{mt}, sk^{mt}). Then it computes a new instance of the multi-time signature scheme, which will be denoted also by (pk^{mt}, sk^{mt}), and sends (r, pk^{mt}) to the home-entity. The home-entity receives (r, pk^{mt}), computes $sk_M \leftarrow sk_M \odot r$, obtains τ (the signature of pk^{mt} with SK_M), chooses a random $r' \in \{0,1\}^s$ and computes $sk_M \leftarrow sk_M \odot (r \oplus r')$. The home-entity returns (τ, r') to the user-entity and deletes r and r'. As in the key generation algorithm, the user-entity sets $r \leftarrow r \oplus r'$ and stores $(r, \tau, pk^{mt}, sk^{mt})$. The home-entity stores sk_M.

The number of periods will be the number of leaves in the Merkle tree. Each period consists of a previously fixed maximal number of signatures instead of time.

The security of the proposed scheme. We have that at any period:

1. the information of the secret key in possession of the home-entity is sk_M, where $sk_M = SK_M \odot \rho_t$, SK_M is the secret key of the instance of the underlying MSS and ρ_t is a random value chosen by the user-entity and the home-entity for the period t.

2. the information of the secret key in possession of the user-entity is $(\rho_t, \tau_T, pk_T^{mt}, sk_T^{mt})$, where τ_t is the signature of pk_T^{mt} by the home-entity, (pk_t^{mt}, sk_t^{mt}) is an instance of the underlying multi-time signature scheme and ρ_t is the random value chosen by the user-entity and the home-entity for the period t.

The MSS is existentially unforgeable under adaptive chosen message attacks. We suppose that the underlying multi-time signature scheme is existentially unforgeable under adaptive chosen message attacks. The proposed construction produces a scheme which is also existentially unforgeable under adaptive chosen message attacks.

Suppose that an intruder obtains the secret of the home-entity at periods $1 \leq \alpha_1 < \cdots < \alpha_a \leq T$ and the secret of the user-entity at periods $1 \leq \beta_1 < \cdots < \beta_b \leq T$, where $\alpha_i \neq \beta_j \ \forall i, j$. We claim that the intruder cannot forge any signature for any period t, where $t \notin \{\beta_1, \ldots, \beta_b\}$. Note that the knowledge of $sk_M = SK_M \odot \rho_t$ without ρ_t does not reveal SK_M for all $t \in \{\alpha_1, \ldots, \alpha_a\}$. On the other hand, $\{(\rho, \tau_{\beta_j}, pk_{\beta_j}^{mt}, sk_{\beta_j}^{mt})\}_{j=1}^{b}$ consists of Merkle signatures and independent instances of the underlying multi-time signature scheme.

If both secrets from the home-entity and the user-entity are compromised at a same period, then the key SK_M is known. Recall that our version of the MSS is forward secure and therefore our resulting scheme remains forward secure.

6 Conclusion

We presented a forward secure version of the MSS. We showed some estimates of the efficiency of the first improvement to the MSS. We presented a signature scheme similar to the intrusion-resilient ones. Instance of the MSS can be used in this new scheme as part of the multi-time signature scheme. In this case, each instance of the MSS (one for the home-entity and another for the user-entity) can employ the parameter $N = 20$ which allow 2^{40} as maximum number of possible signatures with a fast update process and a quite efficient signature calculation.

7 Acknowledgments

We would like to thank Johannes Buchmann, Evangelos Karatsiolis, Christoph Ludwig, and Ulrich Vollmer for helpful comments and fruitful discussions.

References

[BY03] Mihir Bellare and Bennet Yee. Forward-security in private-key cryptography. In M. Joye, editor, *Topics in Cryptology - CT-RSA'03*, Lectures Notes in Computer Science. Springer-Verlag, 2003.

[ISA] ISAAC. Indirection, shift, accumulate, add, and count. http://www.burtleburtle.net/bob/-rand/isaacafa.html.

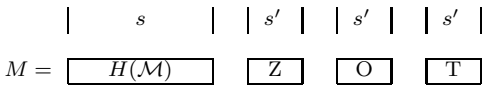
[Mer90] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology - CRYPTO'89 LNCS*, volume 435. Springer-Verlag Berlin Heidelberg 1990, 1990.

[Ope] OpenSSL. Openssl project. http://www.openssl.org/.

A An improved Lamport-Diffie one-time Signature scheme

A improved version of the Lamport-Diffie one-time signature is given by Merkle in [Mer90].

In our experiments we have implemented our improved version of the Lamport-Diffie one-time signature scheme. Suppose that $H : \{0,1\}^* \to \{0,1\}^s$ is a hash function and let s'' be an integer such that $s'' = \lceil \frac{s+3\lfloor \log_2 s \rfloor}{2} \rceil$



$M = H(\mathcal{M})\|Z\|O\|T$, $s' = \lfloor \log_2 s \rfloor$.

Let $\mathcal{M}$ be a message. We represent $H(\mathcal{M})$ as quits (quaternary digits) and let Z, O and T be the corresponding bit string representation of the quantity of zeros, ones and twos in the representation of $H(\mathcal{M})$ as quits, respectively.

Deterministic key generator algorithm *Gen*. On input s and $seed \in \{0,1\}^s$, *Gen* sets $g_{-1} \leftarrow seed$ and computes $(g_i, x_i) \leftarrow prg(g_{i-1})$ $0 \le i < s''$. *Gen* outputs $g = seed$ as the private key and $Y = H(H^3(x_0), \ldots, H^3(x_{s''-1}))$ as the verifying key.

Signature algorithm *Sig*. On input a message $\mathcal{M}$ and the private key g, *Sig* computes $M = m_0 \cdot \ldots \cdot m_{s''-1}$ as quits from $\mathcal{M}$. *Sig* outputs the signature $\tau = (H^{m_0}(x_0), \ldots, H^{m_{s''-1}}(x_{s''-1}))$, where x_i is computed from g as in *Gen* for $0 \le i < s''$.

Verification algorithm Ver. On input a message $\mathcal{M}$, a signature τ' and a public key Y, Ver computes $M = m_0 \cdot \ldots \cdot m_{s''-1}$ as quits from $\mathcal{M}$. Suppose that $\tau' = (z_0, \ldots, z_{s''-1})$. Ver outputs $true$ if $Y = H(H^{3-m_0}(z_0), \ldots, H^{3-m_{s''-1}}(z_{s''-1}))$ and $false$ otherwise.