Reviewing and improving digital signature schemas

Ruohan Zhang

Zhengzhou Foreign Language School, China

Keywords: Digital signature, Lamport one-time digital signature, Batch signature schema, N-ary tree schema.

Abstract: A digital signature scheme offers a cryptographic analogue of handwritten signatures that, in fact, provides much stronger security guarantees. Digital signatures serve as a powerful tool and are now accepted as legally binding in many countries; they can be used for certifying contracts or notarizing documents, for authentication of individuals or corporations, and as components of more complex protocols. Digital signatures also enable the secure distribution and transmission of public keys and thus, in a very real sense, serve as the foundation for all of public-key cryptography.[1] The paper introduces the representative examples of digital signature schemes: Lamport one-time digital signature, Batch signature schema and N-ary tree schema. Each schema with detailed mathematical explanations and analysis of its advantages and defects are presented below.

1. Introduction

Digital Signature is a device which helps you to verify the message you receive is exactly from the one you want to hear from and the message itself has not been changed by someone else. In other words, digital signature helps secure the message's authentication and integrity; thus, make the exact message convey between two exact people. Its characteristics make it especially useful in business field and during war time.

The signer - the sender - first need to use an algorithm to generate a private key and a matching public key. Then it should keep the private key to their own, but can make the public key entire public. When uses it to sign messages, the first need to hash the original message according to the selected scheme, and then sign that message with the private key. The versifier - the receiver - needs to use the matching public key to later verify. And since the public key are made public, actually everyone can verify the message, but with the public key no one can sign any new messages.

2. Lamport one-time digital signature

One-time signature schema, a kind of digital signature schemas, is used to sign at most one message; otherwise the signature can be forged [2]. One of the advantages is that the one-time signature generation and verification are very efficient and it is useful for chip cards, where low computation complexity is required. Lamport first invented a one-time digital signature schema based on one-way functions.[3] The Lamport One-Time Signature Schema (LOTSS) is a signature schema in which the public key can only be used to sign a single message. The security of the LOTSS is based on cryptographic hash functions. Any secure hash function can be used, which makes this signature schema very adjustable. If a hash function becomes insecure it can easily be exchanged by another secure hash function.[4] In this section, we briefly review the Lamport one-time digital signature, which includes three attempts with three algorithms: key generation, signature and verification.

2.1 Lamport one-time digital signature first attempt

2.1.1 Key generation

Using OWF, denote f.

Message=m [n bits], SK = (x1, x2...xn), PK(f(x1), f(x2) ...f(xn)).

2.1.2 Signature

If m[i] = 1, xi will be part of S. Else if m[i] = 0, xi will not be part of S.

The signature would contain the number of 1s in the message, and define it as length l.

The time for signing would be the speed your computer identify 0 and 1 in the message and pick out corresponding xn in SK.

2.1.3 Verification

Given m, see which of the random numbers should be part of S. Check that those random numbers are correct by apply f and comparing with the value in PK.

Define the time to compute a one-way hash function to be t_f. The time for verifying is l*t_f.

2.2 Lamport one-time digital signature second attempt

2.2.1 Key generation

Using OWF, denote f.

Message=m [n bits], SK= (x1, x2...xn),(x1', x2'...xn')[2n bits], PK(f(x1), f(x2) ...f(xn)), (f(x1'), f(x2') ...f(xn')) [2n bits]

2.2.2 Signature

If m[i] = 1, xi' will be part of S. Else if m[i] = 0, xi will be part of S.

The signature would be the same length as the message, and define it as length l.

The time for signing would be the speed your computer identify 0 and 1 in the message and pick out corresponding xn or xn' in SK.

2.2.3 Verification

Given m, see which of the random numbers should be part of S. Check that those random numbers are correct by apply f and comparing with the value in PK.

Define the time to compute a one-way hash function to be t_f. The time for verifying is l*t_f.

2.3 Lamport one-time digital signature improved attempt

2.3.1 Key generation

Using OWF, denote f.

Message=m [n bits], SK= (x1, x2...xn)[n+logn(base 2) +1 bits], PK(f(x1), f(x2) ...f(xn))[n+logn(base 2) +1 bit]

2.3.2 Signature

m || Z(m), If m[i] = 1, xi will be part of S. Else if m[i] = 0, xi will not be part of S.

The length of the signature equals the number of 1s in m||Z(m), and define it as l.

The time for signing would be the speed your computer identify 0 and 1 in m||Z(m) and pick out corresponding xn in SK.

2.3.3 Verification

Given m, see which of the random numbers should be part of S. Check that those random numbers are correct by apply f and comparing with the value in PK.

Define the time to compute a one-way hash function to be t_f. The time for verifying is l*t_f.

2.4 Lamport one-time digital signature schema conclusion

In the above case, 'The first Attempt' is the simplest scheme, but it is insecure since any one can sign another message with a known signature; the second one is secure, but it can be only used once, and the key would be too long; the third one is most recommended in this case, for it is not only safe, but also comparably short, but it also can only be used once. To sum up, the whole case is not ideal and efficient enough.

3. Batch signature schema

The concept of batch signing multiple digital signatures is to find a method by which multiple digital signatures can be signed simultaneously in a lower time complexity than separately signing all the signatures.

In the below case, define the length of S to be l, and since the verifying time of S is comparably small to the hashing process, we just ignore it in the following calculation. And we define the time used to calculate a hash function as t_h, the length of the hash function as l_h.

3.1 Batch signature first attempt

3.1.1 Key generation

Using OWF and Collision-Resistant hash function, denote f and h.

Message=m, m2...m_k[n bits each], SK= (x1, x2...xn) PK(f(x1), f(x2) ...f(xn)).

3.1.2 Signature

h (m1 || m2 || .....|| m_k), signature on m1 = S, (m2, m3, ... m_k) = S1, signature on m2 = S, (m1, m3, ... m_k) = S2...

The length of each signature would be (k-1)m+l.

The time for signing would be 1*t_h.

3.1.3 Verification

Given m1 and S1, recover all messages. Compute h (m1, m2, ... m_k), S is part of S1. Verify that S is indeed a signature on h (m1, m2, ... m_k).

The time for verifying is 1*t_h.

3.2 Batch signature second attempt

3.2.1 Key generation

Using OWF and Collision-Resistant hash function, denote f and h.

Message=m, m2...m_k[n bits each], SK= (x1, x2...xn), PK(f(x1), f(x2) ...f(xn)).

3.2.2 Signature

Sign: H = h(H1, H2), H1 = h(m1 || m2 || .....|| m_k/2), H2 = h(mk/2+1 || | .....|| m_k)... Signature on m1 = S, (m2, m3, ... m_k/2) , H2 = S1 signature on m2 = S, (m1, m3, ... m_k/2) , H2 = S2...

The length of each signature would be l+(k/2-1)m+l_h.

The time for signing would be 3*t_h.

3.2.3 Verification

Given m1 and S1, recover the first half messages. Compute H1 = h (m1 || m2 || .....|| m_k/2). Then compute H = h (H1, H2). Verify that S is indeed a signature on H = h (H1, H2).

The time for verifying is 2*t_h.

3.3 Batch signature improved attempt

3.3.1 Key generation

Using OWF and Collision-Resistant hash function, denote f and h.

Message=m, m2...m_k[m bits each], SK= (x1, x2...xn), PK(f(x1), f(x2) ...f(xn)).

3.3.2 Signature

H=h(H1,H2...H_n), H1 = h(m1||m2||...m-k/n), H2 = h(mk/n+1||...m-k*2/n) ...Hn, signature on m1 = S, (m2, m3, ... m_k/n), H2,H3...Hn, ... ,signature on mk = S, H1,H2....Hn-1, (m-k(n-1)/n+1,.... m-k/n-1).

The length of each signature would be l+(n-1)l_h+(k/n-1) m.

The time for signing would be (n+1)*t_h.

3.3.3 Verification

Given m1 and S1, recover the first 1/n part of the messages. Compute H1 = h (m1||m2||...m-k/n). Then compute H=h (H1, H2...H_n). Verify that S is indeed a signature on H = h (H1, H2).

The time for verifying is 2*t_h.

3.4 Batch signature schema conclusion

In the above case, all of the attempts are secure. However, the second and the third ones are more preferable due to their shorter signature on each message. The second and the third one is essentially based on the same idea, but the third one divides all of the messages into more parts which can make the signature much shorter.

4. N–ary tree schema

In this case define the length of S to be l again, and since the verifying time of S is comparably small to the hashing process, we just ignore it in the following calculation. And we define the time used to calculate a hash function as t_h, the length of the hash function as l_h.

4.1 Binary trees

4.1.1 Key generation

Assume that k is a power of 2 = 2^m.

Using OWF and Collision-Resistant hash function, denote f and h.

Message=m, m2...m_k[m bits each], SK= (x1, x2...xn)[l_h bits], PK(f(x1), f(x2) ...f(xn))[l_h bits].

4.1.2 Signature

The length of the signature would be m + l_h*logk+l.

The time for signing would be (k + k/2+k/4+...+1)*t_h.

4.1.3 Verification

Follow the trees and finally compute H1k = h (H1k/2, Hk/2k). Verify that S is indeed a signature on H1k = h (H1k/2, Hk/2k).

The time for verifying is (logk+1)*t_h.

4.2 3-ary trees

4.2.1 Key generation

Assume that k is a power of 3 = 3^m.

Using OWF and Collision-Resistant hash function, denote f and h.

Message=m, m2...m_k[m bits each], SK= (x1, x2...xn)[l_h bits], PK(f(x1), f(x2) ...f(xn))[l_h bits].

4.2.2 Signature

The length of the signature would be m + l_h*2*logk(base 3) +l.

The time for signing would be (k + k/3+k/9+...+1)*t_h.

4.2.3 Verification

Follow the trees and finally compute H1k = h (H1k/3, Hk/3 2k/3, H2k/3 k). Verify that S is indeed a signature on H1k = h (H1k/3, Hk/3 2k/3, H2k/3 k).

The time for verifying is t_h*(logk+1).

4.3 5-ary trees

4.3.1 Key generation

Assume that k is a power of 5 = 5^m.

Using OWF and Collision-Resistant hash function, denote f and h.
Message=m, m2...m_k[m bits each], SK= (x1, x2...xn)[l_h bits], PK(f(x1), f(x2) ...f(xn))[l_h bits].

4.3.2 Signature

The length of the signature would be m+l_h*4*logk(base 5) +l.
The time for signing would be (k+k/5+k/25+...+1)*t_h.

4.3.3 Verification

Follow the trees and finally compute H1k = h (H1k/5, Hk/5 2k/5, H2k/5 3k/5, H3k/5 4k/5, H4k/5 k). Verify that S is indeed a signature on H1k = h (H1k/5, Hk/5 2k/5, H2k/5 3k/5, H3k/5 4k/5, H4k/5 k).

The time for verifying is t_h*(logk+1)

Table1: The time for verifying

	Binary trees	3-ary trees	5-nary trees
Signing time	(k+k/2+k/4+...+1) *t_h	(k+k/3+k/9+...+1) *t_h	(k+k/5+k/25+...+1) *t_h
Verifying time	t_h*(logk+1)	t_h*(logk+1)	t_h*(logk+1)

4.4 N-ary trees schema conclusion

From the Table1 we can know that for the schemes in this category, the more messages you decide to divide each time, the faster you are able to sign and verify the whole bunch of messages.

5. Conclusion

We have proposed the generalized Lamport one-time signature schema which saves storage space, and batch signature with n-ary trees signature. It is summarized that each schema with detailed mathematical explanations and analysis of its advantages and defects. We expect that the attempted and improved schemas can be used to build more operative signature schemas.

References

[1] Jonathan Katz, Digital signatures: Backgroung and definations, 10.1007/978-0-038-27712-7_1

[2] M. Bellare, S. Micali, How to sign given any trapdoor function, Journal of Cryptology-Crypto'92, LNCS 740 (1993) 1-14

[3] Ming-Hsin Chang, Yi-Shiung Yeh, Improving Lamport one-time signature schema, Institute of Computer Science and Information Engineer

[4] Georg Becker, Merkle Signature Schemas, Merkle Trees and their Cryptanalysis, Seminararbeit Ruhr-University Bochum