Universally Composable Key-evolving Signature

Jin Zhou [1], TingMao Chang [1], YaJuan Zhang[1,2], YueFei Zhu[1]

[1] Network Engineering Department Information Engineering University, Zhengzhou 450002, Henan, China
[2] Key Laboratory of Information Engineering, Guangzhou University, Guangzhou 510006, Henan, China
(Jin Zhou, zhoujin820916.jojo@yahoo.com.cn)

Abstract. The standard digital signature scheme can be easily subject to key exposure problem In order to overcome this problem; a feasible and effective approach is employed by key-evolving signature scheme. In this paper, we study key- evolving signature within the UC framework and propose an appropriate ideal functionality that captures the basic security requirements of key-evolving signature. Then, we present a generic way to transform a key-evolving signature scheme into a real-life protocol. Finally, we show that UC definition of security is equivalent to previous definition of security which is termed as EU-CMA security.

Keywords: Digital signature, Universally composable, Ideal functionality, Forward secure, Key-evolving, EU-CMA

1 Introduction

Digital signature, which was firstly proposed by Diffie and Hellman in [1], plays a very important role in the modern cryptography. An intuitive and widely accepted formalization of the security requirements of signature scheme, called EU-CMA security, was first put forth in [1]. EU-CMA, namely existential unforgeability against chosen message attacks, is also an accepted definition of security of other non-standard digital signature schemes, such as key-evolving signature scheme described below.

The standard digital signature scheme can be easily suffered from key exposure problem, which has been classified as one of the biggest problems for a security system. The system security is completely compromised once the key is exposed. To address this problem, several different approaches have been suggested. Many of them try to minimize exposure of the secret by splitting it and storing the parts in different places, usually via secret sharing [2, 3]. However, as indicated in [4], those approaches can be quite costly and not a viable option for the user.

A feasible approach is employed by forward-secure digital signature scheme, which was firstly formalized by Bellare and Miner in [4]. Following the initial work by [4], a sequence of other deviations of the forward-secure signatures was proposed [5, 6, 7, 8]. All these forward-secure signature schemes involve updating the secret

key periodically. Therefore, a forward-secure signature scheme is, first, a key-evolving signature scheme. Forward security results from the fact that the update algorithm is a one-way function and it is very difficult for an adversary to recover previous secret keys even if the secret key in the current time-period is known.

A general framework for representing cryptographic protocols and analyzing their security is presented by R.Canetti [9, 10]. This framework allows defining the security properties of practically cryptographic tasks. Most importantly, in this framework security of protocols is preserved under a very general composition operation with an unbounded number of copies of arbitrary protocols running concurrently in the system. This composition operation is called universal composition. Similarly, definitions of security formulated in this framework are called universally composable (UC) security.

The definitions in the UC framework follow a definitional approach which is referred to as "security by emulation of an ideal process". In the last few years, research on the relation between emulation-based definition of security and conventional definition of security has become one of the significant topics in cryptography [11]. One case where the conventional definition and emulation-based definition of security were shown to be equivalent is semantically secure encryption against adaptive chosen ciphertext attack [10].

In this paper, we study and formulate an appropriate ideal functionality, which captures the basic idea and security requirements of key-evolving signature. Then, we propose that how to convert a generic key-evolving signature scheme $\sum$ into a real-life protocol $\pi_{\sum}$. Finally, we prove that $\sum$ is EU-CMA if and only if $\pi_{\sum}$ UC-securely realizes our proposed ideal functionality.

2 Preliminary

We sketch a key-evolving digital signature scheme in section 2.1, and recall some useful notions within the UC framework in section 2.2.

2.1 Key-evolving Digital Signature Schemes

A generic key-evolving digital signature scheme $\sum$ is an algorithm with the quadruple, $\sum$ = (*Gen*,*Upd*,*Sig*,*Ver*), such that:

1. *Gen*: the key generation algorithm is a probabilistic algorithm that inputs a security parameter $k \in N$ (given in unary as 1^k) and the total number of time-periods T, and returns a public key PK and the initial secret key SK_0.
2. *Upd*: the secret key update algorithm, accepts as input the secret key SK_i for the current time-period, and returns the new secret key SK_{i+1} for the next time-period. This algorithm is usually deterministic.

3. *Sig*: the signing algorithm, accepts as input the secret key SK_i in the current time period and a message *m*. It returns a pair (*i,sign*), that represents the signature of *m* and time-period index *i*. This algorithm may be probabilistic.

4. *Ver*: the verification algorithm, accepts as input the public key *PK*, a message *m* and a candidate signature (*i,sign*), and returns 1 if the signature of *m* is valid and returns 0 otherwise. This algorithm is typically deterministic.

The verification algorithm is required to verify that whether a signature of *m* generated via $Sig(SK_i, m)$ is valid for period *i*. For convenience, it is also assumed that the secret key SK_i for time-period $i \in \{0,\ldots,T-1\}$, always contains both the value *i* and the total number of periods T . For the last time period $T-1$, $Upd(SK_{T-1})$ returns the empty string SK_T.

2.2 Some Useful Notions in the UC Framework

Recall that there are three types of messages, which are input message, output message and incoming message, transmitted in the UC framework. (The detail of these three types of messages is in [10].)

Dummy adversary. Dummy adversary, denote $\tilde{D}$, is a special type of adversaries. It proceeds as follows:

1. When activated with an input *x* which contains identity of some party from *Z*, it delivers this message to relevant party.
2. When activated with an incoming message *m* from some party, it passes *m* as output to *Z*.
3. It corrupts parties when instructed by *Z*, and passes all gathered messages to *Z*.

Note that a real-life protocol UC-securely realizes the ideal functionality if and only if it UC-securely realizes the ideal functionality with respect to dummy adversary [10].

Party corruptions. Adaptive party corruptions, namely corruptions that occur as the computation proceeds, based on the information gathered by the adversary so far. Arguably, adaptive corruption of parties is a realistic threat in existing networks. Nonetheless, it is sometimes useful to consider also a weaker threat model, called non-adaptive party corruptions, where the identities of the adversarially controlled parties are fixed before the computation starts.

Active party corruptions mean that adversary obtains total control over the behavior of corrupted parties. Another standard corruption model assumes that even corrupted parties continue to follow their prescribed protocol. Here the only advantage the adversary gains from corrupting parties is in learning the internal states of those parties. Such party corruption is called passive party corruption.

3 Definition of Ideal Functionality

In this section, we formulate UC definition of security for key-evolving signature.

The basic idea of $F_{KE\text{-}SIG}$ is to provide a "registry service". The signer S can register (*message, signature*) pairs. Any party that provides the correct verification algorithm can check whether a given pair is registered.

Functionality $F_{KE\text{-}SIG}$ is presented in Figure 1. As expected, it begins with **KeyGeneration** phase. Upon receiving KeyGeneration message from party S, $F_{KE\text{-}SIG}$ asks the ideal process adversary to provide the tuple (u,s,v,SK_0): a polytime deterministic updating algorithm u, a polytime probabilistic signing algorithm s, a polytime deterministic verification algorithm v and private signing key SK_0.

$F_{KE\text{-}SIG}$ lets the adversary determine the values of the verification algorithm and the legitimate signature. This reflects the fact that the intuitive notion of security of signature schemes does not make any requirements on these values.

Then $F_{KE\text{-}SIG}$ outputs verification algorithm v to all the parties. In addition, $F_{KE\text{-}SIG}$ initializes variable i=0 to record current time-period and sets a variable f_0 to record whether the signer S is corrupted in the time-period 0.

$F_{KE\text{-}SIG}$

KeyGeneration: Upon receiving a query (*KeyGen, sid,S,T*) with total time-period T from party S, send this query to the adversary. Upon receiving (*Algorithms,sid, S,u,s,v, SK*$_0$) from the adversary, where s are descriptions of **PPT ITM**, u and v are descriptions of a deterministic polytime ITM, output (*VerificationAlgorithm,sid,S,v*) to all the parties and adversary. In addition, set i=0 and f_0=***fresh***. If S is corrupted at this period, output SK_0 and reset f_0=***corrupt***.

Update: Upon receiving a query (*Update,sid,S*) from party S (for party S only), let $SK_{i+1}=u(SK_i)$, erase SK_i, set $i \leftarrow i+1$, output i and set f_i=***fresh***. If the signer S is corrupted in the period i, output SK_i and reset f_i=***corrupt***.

Signature: Upon receiving a query (*Sign,sid,S,m*) from party S (for party S only), let $\sigma = (i,sign) = s(SK_i,m)$, and verify that $v(m,\sigma)=1$. If so, output (*Signature,sid S,m,*σ) to S and record the pair (m,σ). Else, output error message and halt.

Verification: Upon any party V receiving a query (*Verify,sid,m',*σ'*,v'*) with $\sigma'=(i',sign')$ from any party V, then do: if $v'=v$, $v'(m',\sigma')=1$, $f_{i'}$=*fresh* and no pair (m',σ'') for any σ'' is recorded, then output an error message to V and halt. Else, output (*Verified,sid,S, m',*σ'*,v'*(m',σ') to S.

Fig 1: The key-evolving signature ideal functionality $F_{KE\text{-}SIG}$

Upon receiving Update message from party S (and for party S only), $F_{KE\text{-}SIG}$ "enters" the next time-period by computing the new private signing key SK_{i+1}, erasing the old private signing key SK_i ,and updating variable $i \leftarrow i+1$ and setting new variable f_i to record whether the signer S is corrupted in the time-period i.

Upon receiving a query from party S (and for party S only) to sign a message m, $F_{KE\text{-}SIG}$ first obtains a σ by running the algorithm s. It then verifies that $v(m,\sigma)=1$. If so, it outputs the signature σ to S and records the pair (m,σ); Else, $F_{KE\text{-}SIG}$ outputs

an error message and halt. Verifying that $v(m, \sigma)=1$ in the Signature phase guarantees ***Completeness***, namely that if a signature was generated "honestly" (i.e. via $F_{KE\text{-}SIG}$) then it will be correctly verified.

In the Verification phase, $F_{KE\text{-}SIG}$ checks if the input $(m', \sigma'=(i', sign'), v')$ consists of a forgery, namely if $v'=v$, $v'(m', \sigma')=1$, $f_{i'}=fresh$ and no pair (m', σ'') for any σ'' is recorded. If so, $F_{KE\text{-}SIG}$ outputs an error message and halt. So ***Unforgeability*** is guaranteed. Else, it outputs $v'(m', \sigma')$. If the verification algorithm v' presented by the verifier is not the registered one (i.e. v), $F_{KE\text{-}SIG}$ provides no guarantee regarding the result of the verification phase. This captures the fact that the basic notion of signature scheme only binds messages and signatures to verification algorithms, rather than the party identities. It's the responsibility of the protocol that invokes $F_{KE\text{-}SIG}$ to make sure that the verification algorithm is correct.

4 EU-CMA Security and Real-life Protocol

In this section we state the security definition of EU-CMA for key-evolving signature scheme in section 4.1 and how to transform a key-evolving signature scheme into a real-life protocol in section 4.2.

4.1 The definition of EU-CMA Security

The definition of EU-CMA security, which is a security requirement for standard signature scheme, was first proposed by in [1]. In this section, we state a little modified variant of definition of EU-CMA security for key-evolving signature scheme.

Definition 1 A key-evolving signature scheme $\sum = (Gen, Upd, Sig, Ver)$ is called EU-CMA if the following properties hold for any negligible function $p()$, and all large enough values of security parameter k:

Completeness: For any message m and any period i, $\Pr[(PK, SK_0) \leftarrow Gen(1^k); \sigma \leftarrow Sig(SK_i, m); \ 0 \leftarrow Ver(PK, m, \sigma)] < p(k)$.

Consistency: For any message m and any period i, the probability that $Gen(1^k)$ generates (PK, SK_0) and $Ver(PK, m, \sigma)$ generates two different outputs in two independent invocations is smaller than $p(k)$.

Unforgeability: For any PPT adversary G (called forger), the probability that G wins the "EU" game is smaller than $p(k)$. The full description of "EU" game described as follows:

In the "EU" game, a forger G knows public key PK, the total number of time-periods T and the current time-period. For a key-evolving signature scheme $\sum =$

(*Gen,Upd,Sig,Ver*), *G* is functioning in three stages: the chosen-message attack (**CMA**) phase, the **break-in** phase, and the **forgery** phase.

In the **CMA** phase, *G* has access to the signing oracle, and can obtain the signature of any message it selects under the current secret key. The **break-in** phase is used to model the possible key exposure caused by an adversary break-in. In such a case, *G* is given the current secret key SK_i. In the final **forgery** phase, *G* outputs its forgery, i.e. a signature message pair.

The adversary *G* is said to win the game if it forges the signature of some "new" message for some time-period prior to the break-in. Here, the term "new" message is used to indicate some message that has never been queried for the signature by the adversary.

4.2 Transform generic scheme into real-life protocol

We describe how to convert a generic key-evolving scheme $\sum$ = (*Gen*,*Upd*,*Sig*,*Ver*) into a real-life protocol $\pi_{\sum}$. The protocol $\pi_{\sum}$ proceeds as follows:

Protocol $\pi_{\sum}$

Generation: When party *S* (signer) receives a query (*KeyGen*,*sid*,*S*,*T*), it runs algorithm $Gen(1^k)$, obtains pair (PK,SK_0), keeps the signing key SK_0 secretly and outputs (*VerificationAlgorithm*,*sid*,*S*,*v*=*ver*(PK,·)) to the adversary and all the parties. In addition, *S* sets i=0.

Update: When party *S* receives a query (*Update*,*sid*,*S*), it computes $SK_{i+1}=Upd(SK_i)$, erase SK_i, set $i \leftarrow i+1$ and output i to all the parties and adversary.

Signature: When party *S* receives a query (*Sign*,*sid*,*S*,*m*), it computes $\sigma=(i,sign)=sig(SK_i,m)$ and outputs (*Signature*,*sid*, *S*,*m*, σ).

Verification: When any party receives a query (*Verify*,*sid*,*m*',σ',*v*'), it outputs (*Verified*,*sid*,*S*,*m*',σ', *v*'(*m*',σ')).

When a party is corrupted, it reveals its internal state, which includes all past signing and verification requests and answers. If signer *S* is corrupted, it also reveals the current private signing key SK_i.

Fig 2: The generic key-evolving signature protocol $\pi_{\sum}$

5 Proof of Security

In this section, we will prove that the definition of EU-CMA security is equivalent to definition of UC security.

Theorem 1 let $\sum$ = (*Gen*,*Upd*,*Sig*,*Ver*) be a key-evolving signature scheme, then $\pi_{\sum}$ securely realizes $F_{KE\text{-}SIG}$ with respect to active and adaptive party corruption if and only if $\sum$ is EU-CMA.

Proof: "$\Rightarrow$" Assume that $\sum$ is not EU-CMA, we show that $\pi_{\sum}$ doesn't realize $F_{KE\text{-}SIG}$ under the active and adaptive party corruption. Recall that the definition of UC security with respect to dummy adversary. We should construct an environment *Z* such that for any ideal process adversary *J, Z* can distinguish whether it interacts with dummy adversary $\tilde{D}$ and protocol $\pi_{\sum}$ in the real world or interacts with ideal process adversary *J* and ideal functionality $F_{KE\text{-}SIG}$ in the ideal process.

In following case 1 and case 2, *Z* outputs 1 if it receives an error message, while in following case 3, Z outputs 0 if it receives an error message. The full construction of *Z* is described as follows:

1. Assume that $\sum$ is not **complete**, i.e., there exists $i \in N$ and a message *m*, such that Prob[$(PK,SK_0) \leftarrow Gen(1^k); \sigma \leftarrow Sig(SK_i,m); 1 \leftarrow Ver(PK,m,\sigma)] < 1\text{-}p(k)$ for infinitely many *k*'s. Then *Z* simply activates party *S* with (*KeyGen*,*sid*,*S*,*T*), followed by inputs (*Update*,*sid*,*S*) *i* times and input (*Sign*,*sid*,*S*,*m*), obtains verification algorithm *v* and signature σ. Next *Z* activates some party *V* with input (*Verify*,*sid*,*m*,σ,*v*) and outputs the returned verification value. Then, *Z* always outputs 1 if it interacts with ideal process, while output 0 with non-negligible probability if it interacts with real world.

2. Assume that $\sum$ is **complete** but not **consistent**. Then *Z* operates similarly except that it actives *V* twice with (*Verify*,*sid*,*m*,σ,*v*)and outputs 1 iff the two answers are the identical. Then again, if it interacts with $F_{KE\text{-}SIG}$ in the ideal process, *Z* always outputs 1 since *Z* receives a deterministic polytime ITM. However, *Z* output 0 with non-negligible probability if it interacts with $\pi_{\sum}$ in the real world.

3. Assume that $\sum$ is both **complete** and **consistent** but not ***unforgeable***. In the other words, there exists a forger *G* such that *G* wins the "EU" game with non-negligible probability. Then *Z* proceeds as above except that *Z* internally runs an instance of *G* and hands it the verification algorithm *v* obtained from *S*. From now on, whenever *G* asks its signing oracle to signed a message *m*, *Z* activates signer *S* with input (*Sign*,*sid*,*S*,*m*) and sends the respond signature σ to *G*. When *G* asks **break-in**, *Z* corrupts signer *S* and sends the current private signing key SK_i to *G*. When *G* finally generates a pair (m', σ'), *Z* proceeds as follows. If *m*' was signed before, then *Z* outputs 0 and halts. Else, *Z* activates some party with input (*Verify*,*sid*,*m*',σ',*v*) and outputs the verification result. It can be easily seen that, when *Z* interacts with $\pi_{\sum}$ in the real world, the *G*'s views be exactly an "EU" game on $\sum$, thus *Z* outputs 1 with non-negligible probability. However, *Z* never output 1 if it interacts with $F_{KE\text{-}SIG}$ in the ideal process.

"$\Leftarrow$" Assume that $\pi_{\sum}$ does not UC-securely realize $F_{KE\text{-}SIG}$, we will show

that $\sum$ is not EU-CMA Using the equivalent notion of security with respect to the dummy adversary, we have that for any ideal process adversary *J*, there exists an environment *Z* that can distinguish whether it interacting with dummy adversary $\tilde{D}$ and protocol $\pi_{\sum}$ in the real world or interacts with ideal process adversary *J* and ideal functionality $F_{KE\text{-}SIG}$ in the ideal process.

Since *Z* succeeds for any *J,* it also succeeds for following "generic" *J*. Then *J* does not interact with *Z* at all, except to corrupt parties. *J* runs $(PK,SK_0) \leftarrow Gen(1^k)$, and sends $(u=Upd(\cdot),s=Sig(\cdot),v=Ver(PK,\cdot),SK_0)$ to $F_{KE\text{-}SIG}$. When *Z* instructs to corrupt party, *J* sends to $F_{KE\text{-}SIG}$ a corruption message, and forwards to *Z* the information provided by $F_{KE\text{-}SIG}$.

Assume that scheme $\sum$ is both **complete** and **consistent** (otherwise, the theorem is proven). We argue that it is not ***unforgeable***, by constructing a successful forger *G*. This is done as follows.

G sets i=0 and runs a simulated instance of *Z,* and simulates for the instance of *Z* the interaction with parties:

1. When *Z* activates some party *S* with input (*KeyGen*,*sid*,*S*,*T*), *G* returns its verification algorithm *v* on be half of *S*.
2. When *Z* asks signer *S* to enter the next time-period, *G* sets $i \leftarrow i+1$ and sends *i* to *Z* on be half of *S*.
3. When *Z* asks signer *S* to sign some message *m*, *G* asks its signing oracle for a signature σ on *m*, and returns σ to *Z* on be half of *S*.
4. When *Z* instructs to corrupt some party, *G* returns all signing and verification message request made by such party, and if be the signer, *G* asks **break-in** to obtain current private signing key SK_i and sends it to *Z*.
5. Whenever *Z* activates some party with input (*Verify*,*sid*,*m*',σ',*v*), *G* checks whether (*m*',σ') a success forgery for "EU" game. If yes, *G* outputs that pair and halt. Else it continues the simulation.

We analyze the success probability of *G*. let *B* denote the event that, in the execution of $\pi_{\sum}$ some party is activated with a verification request (*Verify*,*sid*, *m*',σ',*v*), where the pair (*m*',σ') is a success forgery for "EU" game.

Since $\sum$ is both **complete** and **consistent**, we have that as long as *B* does not occur, *Z*'s view of an interaction with real world is statistically close to its view of an interaction with ideal process. However, *Z* can distinguish real world and ideal process with non-negligible probability. Thus it is guaranteed that, when *Z* interacts with dummy adversary $\tilde{D}$ and protocol $\pi_{\sum}$ in the real world, event *B* occurs with non-negligible probability.

It's easy to see that, from the views of simulated *Z,* the interaction with the forger *G* looks the same as an interaction with $\pi_{\sum}$. This means that *G* wins the "EU" game with non-negligible probability. #

References

1. Needham, R. M., Schroeder, M. D.: Using encryption for authentication in large networks of computers. Communications of the Association for Computing Machinery, 21(21):993 – 999, Dec. 1978
2. Blakley, G. R.: Safeguarding cryptographic keys. Proceedings of AFIPS 1979 National Computer Conference, Vol. 48, pp. 313–317, 1979
3. Shamir, A.: How to share a secret. Communications of ACM, 22(11), pp. 612–613, 1979
4. Bellare, M., Miner, S.: A forward-secure digital signature scheme. Advance in Cryptology – CRYPTO 99 proceedings, Vol. 1666 of Lecture Notes in Computer Science, M. Wiener ed., pp.431–448. Springer- Verlag, 15-19 August 1999
5. Krawczyk, H.: Simple forward-secure signatures for any signature scheme. Proceedings of the7th ACM Conference on Computer and Communications Security, pp. 108–115, ACM Press2000.
6. Itkis, G., Reyzin L.: Forward-secure signatures with optimal signing and verifying. Advances in Cryptology-CRYPTO 2001, Vol. 2139 of Lecture Notes in Computer Science, J.Kilian, ed., pp. 332–354, Springer-Verlag, 2001.
7. Maklin, T., Micciancio, D., Miner, S.: Efficient generic forward-secure signatures with an unbounded number of time periods. Advances in Cryptology – Eurocrypt 2002, Vol. 2332 of Lecture Notes in Computer Science, L. Knudsen ed., pp. 400–417, Springer-Verlag, 2002
8 Fei Hu, Chwan-Hwa Wu, Irwin, J. D.: A New Forward-security Signature Scheme using Bilinear Maps. http://eprint.iacr.org/2003/188
9. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. 42nd IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, pp. 136–145, 2001.
10 Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols 2005. Revision 3 of ECCC Report TR01-016
11 Canetti. R., Rabin, T.: Universal Composition with Joint State. Advances in Cryptology Crypto 2003, LNCS vol. 2729, Springer–Verlag, pp. 265–281, 2003