# Computationally Sound Symbolic Security Reduction Analysis of Group Key Exchange Protocol using Bilinear Pairings

ZHANG Zijian, [1]ZHU Liehuang, LIAO Lejian
Beijing Lab of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing 100081, China

## Abstract

Canetti and Herzog have proposed a universally composable symbolic analysis (UCSA) of mutual authentication and key exchange protocols within universally composable security framework. It is fully automated and computationally sound symbolic analysis. Furthermore, Canetti and Gajek have analyzed Diffie-Hellman based key exchange protocols as an extension of their work. It deals with forward secrecy in case of fully adaptive party corruptions. However, their work only addresses two-party protocols that use public key encryptions, digital signatures and Diffie-Hellman exchange.

We make the following contributions. First, we extend UCSA approach to analyze group key exchange protocols that use bilinear pairings exchange and digital signatures to resist insider attack under fully adaptive party corruptions with respect to forward secrecy. Specifically, we propose an formal algebra, and property of bilinear pairings in the execution of group key exchange protocol among arbitrary number of participants. This provides computationally sound and fully automated analysis. Second, we reduce the security of multiple group key exchange sessions among arbitrary number of participants to the security of a single group key exchange session among three participants. This improves the efficiency of security analysis.

*Keywords:* Universally Composable Symbolic Analysis, Computational Soundness, Bilinear Pairings, Group Key Exchange Protocol, Forward Secrecy.

[1]Corresponding Author.

## 1. Introduction

Computational and symbolic approaches are two major directions to analyze cryptographic protocols in last two decades. Each approach has its advantage and disadvantage. Computational approach [13, 14] is sound, because it applies computational complexity and probability theory to reduce the security of protocol to some cryptographic hardness assumptions, such as discrete logarithm and decision Diffie-Hellman problems, but its proof is hard to mechanize by computer program [11]. Therefore, it is tedious and highly error prone for even moderately complex protocols [15]. In comparison, symbolic approach [16] is amenable to automation because of its explicitly algebraic structure. There are many automated tools for symbolic model. For example, ProVerif is a tool of spi calculus [7], and SMV is a tool of model checking [20]. However, none of the existing tools always terminates in tolerant time when analyzing a moderately complex protocol with multiple sessions directly [12]. Moreover, it is often criticized since its soundness is unclear.

Recently, Canetti et al. [11, 12] propose a universally composable symbolic analysis (UCSA) approach, which realizes a computationally sound and fully automated analysis to prove composable security properties of composable properties preserve security of individual sessions, when the analyzed protocol session is composed with the other sessions [11]. This allows researchers only to deal with a single session, when analyzing an overall system that consists of an unbounded number of sessions [11]. As a result, the time complex of algorithm that is used to analyze protocols via symbolic tools is optimized, compared with analyzing the overall system with multiple sessions directly. However, so far as we know, the UCSA approach only deals with two-party cryptographic protocols. Moreover, cryptographic protocols are currently restricted to use public key encryptions, digital signatures and Diffie-Hellman exchange.

As far as we know, many bilinear pairings such as Weil pairing or Tate pairing on elliptic and hyperelliptic curves have been proposed to build group key exchange protocols. It is necessary to analyze group key exchange protocols that use bilinear pairings exchange. Therefore, we extend the UCSA approach to deal with group key exchange protocols that use bilinear pairings exchange and digital signatures to resist insider attack. To the best of our knowledge, this is the first time to provide computationally sound and fully automated analysis of a group key exchange protocol to prove composable security properties.

## *1.1. Our Contributions*

It is tricky to extend the UCSA of protocols from two participants to arbitrary number of participants in symbolic model, because the formal definition of the appropriate algebra in protocol execution, and the appropriate rules for adversary to derive messages are not explicit, considering that the number of participants in the group key exchange protocol is variable. Furthermore, the construction of the mapping algorithm from computational model to symbolic model is complicated for bilinear pairings, because it is difficult to define the property of bilinear pairings used in group key exchange protocol among arbitrary number of participants. In addition, ProVerif cannot terminate in tolerant time, when the number of participants is scalable. We make the following contributions to solve the problems above:

First, we extend the UCSA approach to analyze group key exchange protocol that use bilinear pairings exchange and digital signatures. This allows researchers only to deal with a single group key exchange session, when analyzing the group key exchange protocol that consists of an unbounded number of sessions. Furthermore, the secure protocol satisfies composable security property.

Second, we reduce the security of a single group key exchange session among arbitrary number of participants to the security of a single session among three participants by mathematical induction. This allows researchers only to deal with a single group key exchange session with three participants, when analyzing a single group key exchange session with arbitrary number of participants.

Third, we illustrate how to apply our approach to design and analyze group key exchange protocols to resist insider attack. Specifically, we first apply the compiler in the work of [22] to design a group key exchange protocol based on a family of Bilinear Burmester-Desmedt protocols parameterized by three integers $\alpha, \beta, \gamma$, each of which is denoted by $(\alpha, \beta, \gamma)$-BBD protocol [23], and prove that the compiled protocol is not secure. Then we revise it to be another protocol, and prove that the revised protocol is secure. According to our approach, it is also secure in universally composable (UC) security framework [10].

In addition, since Katz and Shin [21] have proposed a compiler to translate a secure group key exchange protocol to resist outsider attack into a secure protocol to resist insider attack, we only need to design group key exchange protocols to resist outsider attack in symbolic model.

In summary, compared with the UCSA in the work of [11, 12], our contributions provide a computationally sound and efficient automated analysis of group key exchange protocols based on bilinear pairings to prove composable security properties with respect to forward secrecy.

## *1.2. Related work*

There have already been numerous research efforts to bridge the gap between symbolic and computational views of cryptography. Many cryptographic primitives and protocols have been studied. Abadi and Rogaway [1] first prove the computational soundness of formal encryption without key cycles in the case of type-0 symmetric encryption schemes.

Further work focuses on extending the work of [1]. In the passive settings, Micciancio and Warinschi [25] propose a confusion free encryption scheme, and show that their result is completeness for [1]. Laud and Corin [24] consider composed keys. Adão et al. [3] capture a key dependent message security notion that is secure, even if there are key cycles. Herzog [19] proves computational soundness of public key encryption in the case of IND-CCA2 security. Boneh et al. [8] demonstrate a public key encryption scheme that is secure under key dependent message attacks. Gracia and van Rossum [18] prove computational soundness of hash functions. Bresson et al. [9] show computationally sound expressions of modular exponentiation. However their model just secures against passive adversary. Furthermore, bilinear pairings are not considered in their work. Kremer and Mazaré [23] present a bilinear pairing operation, and analyze Joux tripartite Diffie-Hellman protocol, $(\alpha, \beta, \gamma)$-BBD protocol, TAK-2 and TAK-3 protocols. However, their model is only secure in passive settings as well. Moreover, it provides no composable security guarantees. Abadi and Warinschi [2] define formal expressions that is computationally sound against offline guessing attacks. Adão et al. [4] consider computationally sound formal expression of the encryption function from the point of information theoretic security.

In the active settings, Canetti and Gajek [12] propose universally composable symbolic analysis of Diffie-Hellman based key exchange protocols by extending the work of [11]. They deal with key exchange protocols that use Diffie-Hellman exchange and digital signatures. However, their work only handles two-party cryptographic protocols. Moreover, their work does not consider bilinear pairings. Micciancio and Warinschi [26] propose a computationally sound encryption scheme via trace mapping lemma. Galindo et al. [17] extend the mapping lemma to commitment schemes. Backes and Unruh [6] introduce a symbolically sound zero knowledge proof system. Backes et al. [5] present the idealized cryptographic library $Lib^{ideal}$ for automated proofs of cryptographic protocols. The real protocol is computationally sound, if it simulates $Lib^{ideal}$. However, their approach needs to analyze protocol with multiple sessions. Moreover, the composition theorem with joint state has not been considered.

### *1.3. Organization*

Section 2 recalls the preliminaries about the UCSA approach, the functionality of digital signature, bilinear decision Diffie-Hellman assumption, and $(\alpha, \beta, \gamma)$-BBD protocol. Section 3 introduces the symbolic model. We define symbolic protocol and symbolic traces, model symbolic adversary, and provide symbolic security criterion of group key exchange. Section 4 discusses the computational model. We define computational traces and functionality of bilinear pairings, prove the security of bilinear pairings, and define the functionality of group key exchange. Section 5 describes simple protocols. We define the syntax and semantic of simple protocols that use bilinear pairings exchange and digital signatures. Section 6 proves the computational soundness of symbolic security criterion. We discuss the mapping algorithm from computational traces to symbolic traces, prove the mapping lemma and the computational soundness theorem. Section 7 reduces the number of participants. Section 8 designs and analyze two variants of $(\alpha, \beta, \gamma)$-BBD protocol as the examples to apply our approach. Section 9 draws the conclusion.

## 2. Preliminaries

### *2.1. Universally Composable Symbolic Analysis Approach*

The specific steps of the UCSA approach to analyzing a complex system are as follow [11]:

(1) Divide the complex system that consists of cryptographic protocols into individual sessions, where each session is an execution of a relatively simple protocol.

(2)Translate each resulting protocol into a symbolic protocol via a specification language for a class of protocols that have clear symbolic form, called simple protocol. It serves as a dummy language whose syntax is parsed into computational semantic and symbolic semantic [12].

(3) Analyze a single session of each symbolic protocol to check if it fulfills some symbolic security criterions by some automated tools. If not, an execution of the analyzed session can be found to break some rules of the criterions. Then compose all the initialize, symbolic adversary and honest participant actions occurred in the execution orderly to form a symbolic trace, and output the symbolic trace as the evidence to prove the symbolic protocol does not satisfy the criterions.

(4) Prove each original protocol satisfies composable security property in the computational model, if symbolic protocol fulfills the symbolic security criterions. This is done by proving its inverse negative proposition. More specially, if the original protocol does not achieve the composable security property, a corresponding execution can be found. Then compose all the environment, computational adversary and honest participant actions occurred in the execution orderly to form a computational trace. After that, the UCSA approach introduces a mapping algorithm that can map the computational trace to a symbolic trace except with a negligible probability, which makes the corresponding symbolic protocol does not satisfy the criterions.

(5) Prove each protocol keeps the same security properties in the complex system. This is done by using the composition theorem with joint state in UC security framework.

Since the syntax of simple protocol defines a pair of programs $\Pi = (\Pi_0, \Pi_1)$, the UCSA approach has hitherto only dealt with two-party cryptographic protocols. Moreover, since the syntax of simple protocol only contains public key encryption schemes *encrypt* and *decrypt*, digital signature schemes *sign* and $verify$, and key encapsulation mechanism *encaps* and *decaps*, cryptographic protocols are currently restricted to use public key encryptions, digital signatures and Diffie-Hellman exchange.

## *2.2. The Functionality of Digital Signatures*

The functionality of digital signatures has been proposed in [12]. It provides an ideal digital signature. Formal definition of this functionality is described in Fig 1.

**Functionality $\mathcal{F}_{CERT}$**

$\mathcal{F}_{CERT}$ proceeds as follows, running with security parameter $k$.

1. **Signature Generation:** Upon receiving a value $(Sign, sid, m)$ from some party $S$, check that $sid = (s, sid')$ for some $sid'$. If not, then abort. Else hand $(Setup, sid)$ to the adversary. Upon receiving $(Algorithms, sid, s, v)$ from the adversary, where $s, v$ are descriptions of probabilistic polynomial time (PPT) interactive Turning machines (ITMs), output $(Sign\_Alg, sid, s)$ to $S$. Set $\sigma = s(m)$, record the tuple $(m, \sigma, 1)$ and output $(Signature, sid, m, \sigma)$ to $S$.
2. **Signature Verification:** Upon receiving a value $(Verify, sid, m, \sigma)$ from some party $P$, do: If there is an entry $(m, \sigma, b')$, send $(Verified, sid, m, b')$ to $P$. Else if the signer is not marked corrupted, record the entry $(m, \sigma, 0)$ and return $(Verified, sid, m, 0)$ to $P$. Otherwise, record the entry $(m, \sigma, v(m, \sigma))$ and return $(Verified, sid, m, v(m, \sigma))$ to $P$.
3. **Corruption:** Upon receiving a value $(Corrupt, sid, P)$ from the adversary, where $P$ is a signer or verifier, mark $P$ as corrupted.

Fig 1. The Ideal Functionality of Digital Signatures

## *2.3. Bilinear Pairing*

Here we briefly recall the basic definitions of bilinear pairings [23], including the definition of bilinear map and BDDH security. We start with formal definition of bilinear mapping and bilinear pairings as follow:

**Definition 1** (**Bilinear Mapping**). *Assume that $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of the same prime order $q$, $g_1$ is a generator of $\mathbb{G}_1$. $e$ is defined as a bilinear mapping from $\mathbb{G}_1 \times \mathbb{G}_1$ to $\mathbb{G}_2$, if it satisfies three prosperities as follow:*
*(1)* ***Bilinear:*** *$e(g_1^x, g_1^y) = e(g_1, g_1)^{xy}$ for any $x, y \in \mathbb{Z}_q$.*
*(2)* ***Non-degeneracy:*** *$g_2 = e(g_1, g_1)$ is a generator of $\mathbb{G}_2$.*
*(3)* ***Computable:*** *there exists an efficient algorithm to compute $e(u, v)$ for any $u, v \in \mathbb{G}_1$.*

**Definition 2** (**Bilinear Pairings**). *Given a security parameter $k$, a bilinear pairing instance generator $IG$ is defined as a PPT algorithm that outputs a 5-*

*tuple $(q, \mathbb{G}_1, \mathbb{G}_2, g_1, e)$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of same prime order $q$, $g_1$ is a generator of $\mathbb{G}_1$, $e$ is a bilinear mapping from $\mathbb{G}_1 \times \mathbb{G}_1$ to $\mathbb{G}_2$.*

Next we recall BDDH security [23]. It is used to analyze group key exchange protocols, such as Joux tripartite Diffie-Hellman, $(\alpha, \beta, \gamma)$-BBD protocol, TAK-2 and TAK-3 protocols. The formal definition of BDDH security is as follow:

**Definition 3** (**BDDH Security**). *An instance generator IG satisfies BDDH assumption, if for any PPT adversary $\mathcal{A}$ against BDDH , the advantage of $\mathcal{A}$ is negligible in the security parameter. That is,* $Adv^{BDDH}_{\mathcal{A},IG}(k) =$

$$\left| \begin{array}{ll} Pr[\mathcal{A}(q, g_1^x, g_1^y, g_1^z, g_2^{xyz}) = 1 : & (q, \mathbb{G}_1, \mathbb{G}_2, g_1, e) \leftarrow IG(k); x, y, z \leftarrow \mathbb{Z}_q] \\ -Pr[\mathcal{A}(q, g_1^x, g_1^y, g_1^z, g_2^r) = 1 : & (q, \mathbb{G}_1, \mathbb{G}_2, g_1, e) \leftarrow IG(k); x, y, z, r \leftarrow \mathbb{Z}_q] \end{array} \right|$$

Note that BDDH security shows that the probability to distinguish $g_2^{xyz}$ from $g_2^r$ is negligible.

## *2.4. Bilinear Burmester-Desmedt Protocol*

Here we briefly recall a family of Bilinear Burmester-Desmedt (BBD) Protocols parameterized by three integers $\alpha, \beta, \gamma$, such that $\alpha + \beta + \gamma = 0$ and either $\alpha, \beta$ or $\gamma$ is different from 0 [23]. The instance of the protocol corresponding to $\alpha, \beta, \gamma$ is as follow:

**Protocol Description of BBD Protocol**. *Assume that $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic group of the same prime order $q$ with respective generators $g_1$ and $g_2$. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ such that $e(g_1, g_1) = g_2$. Assume that all the participants have known each other before executing the protocol, and all the indexes are taken modulo $n$. The protocol proceeds among all the participants $\{p_i | i \in [0, n-1]\}$ as follow:*
(1) *Each $p_i$ chooses a random number $r_i \in \mathbb{Z}_q^*$, keeps $r_i$ secret, and broadcasts $Z_i = g_1^{r_i}$.*
(2) *Each $p_i$ broadcasts: $X_i = e((Z_{i-2}, Z_{i-1})^{\alpha r_i}) \cdot e((Z_{i-1}, Z_{i+1})^{\beta r_i}) \cdot e((Z_{i+1}, Z_{i+2})^{\gamma r_i}) = g_2^{\alpha r_{i-2} r_{i-1} r_i + \beta r_{i-1} r_i r_{i+1} + \gamma r_i r_{i+1} r_{i+2}}$.*
(3) *Each $p_i$ computes the session key: $K = g_2^{\sum_{i=0}^{n-1} r_i r_{i+1} r_{i+2}}$.*

In particular, we compare time complexity to analyze the BBD protocols via ProVerif. The experiment is performed on two 2.4 Ghz Intel(R) Xeon(R) processors, with 12 GB of main memory, running Windows 7 Professional sever pack 1. ProVerif spends 5.2 minutes to analyze a BBD protocol with a single session among three participants, while it spends 5.8 minutes to analyze the same protocol with multiple sessions among three participants. Furthermore, ProVerif cannot terminate to analyze the same protocol with a single session among four participants in 12 hours. The last three results of ProVerif after running 12 hours are as follow:

```
2200 rules inserted. The rule base contains 2076 rules. 2965 rules in the queue.
2400 rules inserted. The rule base contains 2276 rules. 3401 rules in the queue.
2600 rules inserted. The rule base contains 2476 rules. 3545 rules in the queue.
```

Note that the rules in the queue still increase in the last three results, this indicates that ProVerif cannot terminate in another few hours. As a result, this experiment shows that ProVerif cannot terminate in tolerant time, even it analyzes a BDD protocol with a single session among four participants.

Since only a single BBD session among three participants need to be analyzed, if our approach is used to analyze a BBD protocol with multiple sessions among arbitrary number of participants, our work is necessary to analyze group key exchange protocols using bilinear pairings.

## 3. The Symbolic Model

In this section, we introduce the symbolic model of cryptographic protocols based on bilinear pairings and digital signatures, symbolic adversary, and security criterion of group key exchange.

### *3.1. Symbolic Protocol*

We start to define a protocol algebra that use bilinear pairings exchange and digital signatures. The algebra defines the space of atomic messages and operations to compose the input, outgoing messages, incoming messages and local outputs of the protocol. Here we assume that there exists a trustworthy third party that certifies the signature keys, and all participants and adversary have known all the verification keys before protocol execution. Messages are authenticated by participants via their valid signatures respectively. Formal definition of protocol algebra is as follow:

**Definition 4** (**Protocol Algebra**). *All the messages of a group key exchange protocol use bilinear pairings and digital signatures are assumed to be elements of an algebra $A$. There are nine types of atomic messages:*

(1) *Session Identity Symbols: Session identities $SID$ are used to identify the sessions of the same participants. They are denoted as $\{s_i | i \in \mathbb{N}\}$. We*

*assume that $SID$ are publicly known, and their number is unbounded in the algebra that may change from run to run.*

(2) *Participant Identity Symbols: Participant identities $PID$ are used to identify the participants. They are denoted as $p_1, p_2, \ldots, p_n$. We assume that $PID$ are publicly known, and their number is finite in the algebra.*

(3) *Private and Public Key Symbols: Private keys $Pri$ and public keys $Pub$ are generated temporarily to compute session key in different sessions. The former are denoted as $\{pri_i | i \in \mathbb{N}\}$, while the latter are denoted as $\{pub_i | i \in \mathbb{N}\}$. We assume that the number of $Pri$ and $Pub$ is unbounded in the algebra.*

(4) *Assistant Key Symbols: Assistant keys $AK$ are generated temporarily to compute session key in different sessions. They are denoted as $\{ak_i | i \in \mathbb{N}\}$. We assume that the number of $AK$ is unbounded in the algebra.*

(5) *Signature and Verification Key Symbols: Signature keys $SK$ are used to sign messages. They are denoted as $sk_1, sk_2, \ldots, sk_n$. Verification keys $VK$ are used to verify signature of messages. They are denoted as $vk_1, vk_2, \ldots, vk_n$. We assume that the number of signature and verification keys is finite in the algebra.*

(6) *Random Number Symbols: Random Numbers $R$ are used to generate session identity, private key, and keep the freshness of messages. They are denoted as $\{r_i | i \in \mathbb{N}\}$. We assume that the number of $R$ is unbounded in the algebra.*

(7) *Output Symbols: Outputs $\{Establish,\ Key\}$ are used to indicate the s-tart and end of protocol execution. The former indicates the start of an execution, while the latter indicates the end of an execution.*

(8) *Evaluation Symbols: A true evaluation is denoted as $\top$, while a false evaluation is denoted as $\bot$ that leads to the termination of protocol execution.*

(9) *Garbage and Empty Symbols: Garbage message is denoted as $\varrho$, and empty message is denoted as $\xi$.*

*Furthermore, There are seven types of functions to compose atomic messages:*

(1) *Pair Function. $Pair(m_1, m_2) : A \times A \rightarrow A$. This function is used to generate a 2-tuple of two messages in $A$.*

(2) *Setup Function. $Setup() : \{\} \rightarrow Pri \times Pub$. This function is used to generate a private key and the corresponding public key.*

(3) *Assistant Key Generate Functions.* $AK\_Gen_1(pri, m), AK\_Gen_2(pri, m), \ldots, AK\_Gen_n(pri, m) : PRI \times A \rightarrow AK$. *These function are used to compute the assistant recovery keys via private key and bilinear mapping recursively. We assume that the number of these functions is finite in the algebra.*
(4) *Bilinear Pairing Key Generate Function.* $BP\_Gen(pri, m) : Pri \times A \rightarrow A$. *This function is used to compute a bilinear pairing key.*
(5) *Separate Function.* $Sep(m) : A \rightarrow A \times A$. *This function is used to separate a message in* $A$.
(6) *Signature Function.* $Sign(sk, m) : SK \times A \rightarrow A$. *This function is used to sign messages in* $A$.
(7) *Verify Function.* $Verify(vk, m, \sigma) : VK \times A \times A \rightarrow \{\top, \bot\}$. *This function is used to verify the signature of messages are valid or not.*

Note that each element of the algebra $A$ has a unique representation. That is, each compound message is constructed in a unique way. In addition, the entire session identity of each participant is denoted as $(s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$.
**Example:** Given a 6-tuple $(q, \mathbb{G}_1, \mathbb{G}_2, E, e_1, e_2)$, we take $(\alpha, \beta, \gamma)$-BBD protocol as an example to illustrate the protocol algebra $A$:

(1) Each participant $p_i$ first uses *Setup* to generate a private key and the corresponding public key. More specially, first, a random number $r_i$ is chosen as the private key. Then compute $g_1^{r_i}$ as the corresponding public key. That is, $(r_i, g_1^{r_i}) = Setup()$. Finally, broadcast $Z_i = g_1^{r_i}$.
(2) After receiving $Z_{i-2}$, $Z_{i-1}$, $Z_{i+1}$ and $Z_{i+2}$, the assistant key is computed and broadcasted:
$X_i = AK\_Gen_1(r_i, Pair(Pair(Pair(Z_{i-2}, Z_{i-1}), Z_{i+1}), Z_{i+2}))$
$= g_2^{\alpha r_{i-2} r_{i-1} r_i + \beta r_{i-1} r_i r_{i+1} + \gamma r_i r_{i+1} r_{i+2}}$
(3) After receiving $\{X_1, \ldots, X_n\} \backslash X_i$, the bilinear paring key is computed as follow:
$K = BP\_Gen(r_i, Pair(Pair(\ldots Pair(Pair(X_1, X_2), X_3) \ldots), X_n))$
$= g_2^{\sum_{i=1}^{n} r_i r_{i+1} r_{i+2}}$

Since cryptographic protocols are interactive process, we use a state transition system to define the symbolic protocol. Formal definition of the symbolic protocol in algebra $A$ is as follow:

**Definition 5** (**Symbolic Protocol**). *A symbolic protocol* $\overline{\pi}$ *is a mapping from a set* $S = (A)^*$ *of states, a set* $SID$ *of session identities, a set* $PID$

*of participant identities, a set $\{PID\}$ of the partners, and an element in $A$ which represents possible incoming messages, to a set $\{\top, \bot\}$ of evaluation, or a tuple (“message” $\times$ $MESSAGE$) , or a tuple of (“erase” $\times$ $ERASE$), or a tuple of (“output” $\times$ $OUTPUT$), and a new state $S$. That is:* $\overline{\pi} : S \times SID \times PID \times \{PID\} \times A \rightarrow \{\top, \bot\} \cup (\text{“message”} \times MESSAGE) \cup (\text{“erase”} \times ERASE) \cup (\text{“output”} \times OUTPUT) \times S$

Here $MESSAGE = A \times SID \times PID \times \{PID\}$ represents possible outgoing messages. $ERASE = SID \times PID \times \{PID\}$ represents the erasure of the private key for the specific session. $OUTPUT = \{Establish, Key\} \times \{\xi \cup A\} \times SID \times PID \times \{PID\})$ represents possible outputs.

## *3.2. Symbolic Adversary*

We first discuss what knowledge symbolic adversary can derive from messages obtained in the execution of a symbolic protocol. Since symbolic adversary can read, fake, modify, delete messages or corrupt some honest participants, symbolic adversary is possible to derive messages from long term keys and internal state [13], besides the outgoing messages when a symbolic protocol is executed.

We consider the set of adversarial knowledge as a closure that can be defined recursively. Formal definition of adversarial closure is as follow:

**Definition 6** (**Adversarial Closure**). *Given $m_{Adv}, (m_{Adv} \in A)$, adversary closure $C_{Adv}[m_{Adv}]$ is the smallest set of protocol algebra $A$ that consists of the initial and derivative knowledge sets:*

(1) *Initial Knowledge Set*
*All the session identities.*
*All the participant identities.*
*All the public keys.*
*The private keys of corrupted participants.*
*The signature keys of corrupted participants.*
*All the verification keys.*
*Random numbers generated by adversary itself.*

(2) *Derivative Knowledge Set*
$m_{Adv} \in C_{Adv}[m_{Adv}]$.
$Pair(m_1, m_2) \in C_{Adv}[m_{Adv}]$, *if* $m_1, m_2 \in C_{Adv}[m_{Adv}]$.
$AK\_Gen_i(m_{pri}, m) \in C_{Adv}[m_{Adv}]$, *if* $m_{pri}, m \in C_{Adv}[m_{Adv}]$.
$BP\_Gen(m_{pri}, m) \in C_{Adv}[m_{Adv}]$, *if* $m_{pri}, m \in C_{Adv}[m_{Adv}]$.

*$Sep(m) \in C_{Adv}[m_{Adv}]$, if $m \in C_{Adv}[m_{Adv}]$.*
*$Sign(sk, m) \in C_{Adv}[m_{Adv}]$, if $sk, m \in C_{Adv}[m_{Adv}]$.*
*$Verify(vk, m, \sigma) \in C_{Adv}[m_{Adv}]$, if $vk, m, \sigma \in C_{Adv}[m_{Adv}]$.*

Second, we define how symbolic adversary interacts with symbolic protocols to obtain messages. We define it by the notion of symbolic trace. Informally, symbolic trace indicates the sequence of adversarial actions in the execution of symbolic protocols. Formal definition of symbolic trace is as follow:

**Definition 7** (**Symbolic Trace**). *The symbolic trace $\overline{t}$ of symbolic protocol $\overline{\pi}$ is a sequence of events $H_1, \ldots, H_n$, each of which is one of the triple:*

(1) *Initial Event*
*$[$"input"$, s_i, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i, S_{i,init}]$ describes the input of participant $p_i$ who interacts with participants in $\{p_1, p_2, \ldots, p_n\} \backslash p_i$, and initial state is $S_{i,init}, (S_{i,init} \in S)$ at the start of session $s_i$.*

(2) *Adversary Event*
*$[$"sid"$, m_s]$ represents a session identity $m_s, (m_s \in SID)$.*
*$[$"pid"$, m_p]$ represents a participant identity $m_p, (m_p \in PID)$.*
*$[$"pub"$, m_{pub}]$ represents a public key $m_{pub}, (m_{pub} \in Pub)$.*
*$[$"pri"$, m_{pri}]$ represents a private key $m_{pri}, (m_{pri} \in Pri)$.*
*$[$"ak"$, m_{ak}]$ represents an assistant key $m_{ak}, (m_{ak} \in AK)$.*
*$[$"sk"$, m_{sk}]$ represents a signature key $m_{sk}, (m_{sk} \in SK)$.*
*$[$"vk"$, m_{vk}]$ represents a verification key $m_{vk}, (m_{vk} \in VK)$.*
*$[$"random"$, m_r]$ represents a random number $m_r, (m_r \in R)$.*
*$[$"pair"$, m_1, m_2, m_3]$ represents a message $m_3$ is a pair of $(m_1, m_2)$, where $m_1, m_2, m_3 \in A$.*
*$[$"ak_gen$_i$"$, m_{pri}, m, m_{ak}]$ represents an assistant key $m_{ak}$ is generated by $m_{pri}$ and $m$, where $m_{pri} \in Pri, m \in A, m_{ak} \in AK$.*
*$[$"bp_gen"$, m_{pri}, m, m_{bp}]$ represents a bilinear pairing key $m_{bp}$ is generated by $m_{pri}$ and $m$, where $m_{pri} \in Pri, m \in A, m_{bp} \in A$.*
*$[$"sep"$, m_1, m_2, m_3]$ represents a message $m_1$ is separated to $m_2$ and $m_3$, where $m_1, m_2, m_3 \in A$.*
*$[$"sign"$, m_{sk}, m_1, m_2]$ represents a signature $m_2$ is generated by $m_{sk}$ and $m_1$, where $m_{sk} \in SK, m_1 \in A, m_2 \in A$.*
*$[$"verify"$, m_{vk}, m_1, m_2, m_3]$ represents a signature $m_2$ of message $m_1$ is verified by $m_{vk}$, and the result is $m_3$, where $m_{vk} \in VK, m_1 \in A, m_2 \in A, m_3 \in \{\top, \bot\}$.*

*[“expire”, $m_s, m_p, \{m_{p'}\}$] represents a session $m_s, (m_s \in SID)$ of a participant $m_p, (m_p \in PID)$ with the partners $\{m_{p'}\}$ is expired. Then the adversary receives the private key of the session $m_s$, if it is not empty.*
*[“corrupt”, $m_p$] represents a participant $m_p, (m_p \in PID)$ is corrupted. Then the adversary receives all the states of the corrupted participant $m_p$.*
*[“deliver”, $m, p_i$] represents a message $m, (m \in A)$ is delivered to the participant $p_i$.*

(3) *Honest Participant Event*
*[“erase”, $er$] represents a participant erases the private key of session $er, (er \in A)$.*
*[“message”, $m$], or [“output”, $o$] represents a participant sends a message $m$ or generates a local output $o$.*

Third, we define $\bar{t}$ as a valid trace of symbolic protocol $\overline{\pi}$, if each adversary event is a valid action of an active adversary, and each honest participant event is consistent with the Definition 5. Formal definition of a valid symbolic trace is as follow:

**Definition 8** (**Valid Symbolic Trace**). *Let $\bar{t}$ consist of a sequence events $H_1, \ldots, H_n$. Assume that $C^l_{Adv}$ is an adversarial closure, given all the messages in the first $l^{th}$ events of $\bar{t}$. $\bar{t}$ is a valid symbolic trace, if each $H_i$ satisfies all the conditions below:*

(1) *Initial Event*
*If $H_i =$ [“input”, $s_i, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i, S_{i,init}$], then no previous event of it has occurred for $p_i$.*

(2) *Adversary Event*
*If $H_i =$ [“pair”, $m_1, m_2, m_3$], then $m_1, m_2 \in C^{i-1}_{Adv}$ and $m_3 = Pair(m_1, m_2)$.*
*If $H_i =$ [“ak_gen$_j$”, $m_{pri}, m, m_{ak}$], then $m_{pri}, m \in C^{i-1}_{Adv}$ and $m_{ak} = AK\_Gen_j(m_{pri}, m)$.*
*If $H_i =$ [“bp_gen”, $m_{pri}, m_1, m_2$], then $m_{pri}, m_1 \in C^{i-1}_{Adv}$ and $m_2 = BP\_Gen(m_{pri}, m_1)$.*
*If $H_i =$ [“sep”, $m_1, m_2, m_3$], then $m_1 \in C^{i-1}_{Adv}$ and $(m_2, m_3) = Sep(m_1)$.*
*If $H_i =$ [“sign”, $m_{sk}, m_1, m_2$], then $m_{sk}, m_1 \in C^{i-1}_{Adv}$, and $m_2 = Sign(m_{sk}, m_1)$.*
*If $H_i =$ [“verify”, $m_{vk}, m_1, m_2, m_3$], then $m_{vk}, m_1, m_2 \in C^{i-1}_{Adv}$, and $m_3 = Verify(m_{vk}, m_1, m_2)$.*

(3) *Honest Participant Event*
*If* $H_i = [\text{"}message\text{"}, m]$, $[\text{"}erase\text{"}, er]$, *or* $[\text{"}output\text{"}, o]$, *then* $[\text{"}input\text{"}, s_i, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i, S_{i,init}]$ *occurred in the trace* $\overline{t}$ *before* $H_i$.

At last, we define how a symbolic adversary simulates the execution of a protocol to derive knowledge. we regard it as adversarial strategy. Formal notion of adversarial strategy is as follow:

**Definition 9** (**Adversarial Strategy**). *Adversarial strategy* $\Psi$ *is a sequence of instructions* $I_0, I_1, \ldots, I_n$, *each of which is as follow:*
$[\text{"}receive\text{"}, m]$ $[\text{"}deliver\text{"}, m, p_i]$; $[\text{"}sid\text{"}, m_s]$; $[\text{"}pid\text{"}, m_p]$; $[\text{"}pri\text{"}, m_{pri}]$; $[\text{"}pub\text{"}, m_{pub}]$; $[\text{"}ak\text{"}, m_{ak}]$; $[\text{"}sk\text{"}, m_{sk}]$; $[\text{"}vk\text{"}, m_{vk}]$; $[\text{"}random\text{"}, r]$; $[\text{"}pair\text{"}, m_1, m_2, m_3]$; $[\text{"}ak\_gen_j\text{"}, m_{pri}, m, m_{ak}]$; $[\text{"}bp\_gen\text{"}, m_{pri}, m_{ak}, m_{bp}]$; $[\text{"}sep\text{"}, m_1, m_2, m_3]$; $[\text{"}sign\text{"}, sk, m_1, m_2]$; $[\text{"}verify\text{"}, vk, m_1, m_2, m_3]$; $[\text{"}expire\text{"}, s_i, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i]$; $[\text{"}corrupt\text{"}, p_i]$.

*With the execution of a symbolic protocol* $\overline{\pi}$, $\Psi$ *produces the following symbolic trace* $\Psi(\overline{\pi})$. *Pass through each instruction in* $\Psi$: *(1) For each form of* $[\text{"}receive\text{"}, m]$ *instruction, if* $p_i$ *is the first to be activated, or* $p_i$ *was just activated with a delivered message* $m$, $m$ *is added to* $\Psi(\overline{\pi})$. *Else,output the trace* $\perp$ *for failure. (2) For any other instruction, add the corresponding events to the adversarial trace. If there is an event which results in an invalid symbolic trace, then output* $\perp$.

## *3.3. Symbolic Security Criterion*

We first define *Pat* function that is used to formally specify what knowledge a symbolic adversary can obtain from a symbolic trace. Intuitively, a symbolic adversary can obtain the result of *BP_Gen* function, if the private key is in adversarial closure. Furthermore, a symbolic adversary can obtain the result of *Sign* function, if the signature key is in adversarial closure. Formal definition of the *Pat* function is as follow:

**Definition 10** (**Pat Function**). *Let* $\overline{t}$ *consist of* $H_0, H_1, \ldots, H_n$, $T_{pri} = \{Pri\} \cap C^n_{Adv}$ *and* $T_{sk} = \{SK\} \cap C^n_{Adv}$. $Pat(\overline{t}, T)$ *is defined the same as* $\overline{t}$, *except that for each message* $m, (m \in A)$ *in* $H_i$, $m$ *is replaced by* $Pat(m, T)$ *as follow:*

(1) $Pat(m_s, T) = m_s$, *if* $m_s \in SID$
(2) $Pat(m_p, T) = m_p$, *if* $m_p \in PID$
(3) $Pat(m_{pri}, T) = m_{pri}$, *if* $m_{pri} \in Pri$

(4) $Pat(m_{pub}, T) = m_{pub},\ if\ m_{pub} \in Pub$
(5) $Pat(m_{ak}, T) = m_{ak},\ if\ m_{ak} \in AK$
(6) $Pat(m_{sk}, T) = m_{sk},\ if\ m_{sk} \in SK$
(7) $Pat(m_{vk}, T) = m_{vk},\ if\ m_{vk} \in VK$
(8) $Pat(m_r, T) = m_r,\ if\ m_r \in R$
(9) $Pat(m_o, T) = m_o,\ if\ m_o \in \{Establish, Key\}$
(10) $Pat(m_{eva}, T) = m_{eva},\ if\ m_{eva} \in \{\top, \bot\}$
(11) $Pat(m_{ge}, T) = m_{ge},\ if\ m_{ge} \in \{\varrho, \xi\}$
(12) $Pat(Pair(m_1, m_2), T) = Pair(Pat(m_1, T), Pat(m_2, T))$
(13) $Pat(AK\_Gen_i(m_{pri}, m), T) = AK\_Gen_i(Pat(m_{pri}, T), Pat(m, T))$
(14) $Pat(BP\_Gen(m_{pri}, m), T) = BP\_Gen(Pat(m_{pri}, T), Pat(m, T)),$
$if\ m_{pri} \in T_{pri}$
(15) $Pat(BP\_Gen(m_{pri}, m), T) = BP\_Gen(\mathcal{T}_{pri}, Pat(m, T)),\ if\ m_{pri} \neq T_{pri}$
(16) $Pat(Sep(m), T) = Sep(Pat(m, T))$
(17) $Pat(Sign(m_{sk}, m), T) = Sign(Pat(m_{sk}, T), Pat(m, T)),\ if\ m_{sk} \in T_{sk}$
(18) $Pat(Sign(m_{sk}, m), T) = Sign(\mathcal{T}_{sk}, Pat(m, T)),\ if\ m_{sk} \notin T_{sk}$
(19) $Pat(Verify(m_{vk}, m, \sigma), T) = Verify(Pat(m_{vk}, T), Pat(m, T), Pat(\sigma, T))$

Note that $\mathcal{T}_{pri}$ represents the type tree of $m_{pri}$, while $\mathcal{T}_{sk}$ represents the type tree of $m_{sk}$. We do not use symbol $\Box$ for $BP\_Gen(m_{pri}, m)$ or $Sign(m_{sk})$, when $m_{pri} \notin T_{ak}$ or $m_{sk} \notin T_{sk}$, because $m$ or $m_{sk}$ is publicly known respectively.

Next, we define symbolic criterion of group key exchange as follow:

**Definition 11** (**Symbolic Security Criterion of Group Key Exchange**). *A symbolic protocol $\overline{\pi}$ provides secure group key exchange (SGKE), if*

(1) *Agreement*
*After $[\text{“}output\text{”}, Establish, \xi, s_i, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i]$ and $[\text{“}output\text{”}, Key, \kappa_i, s_i, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i]$ occurred in a symbolic trace for each honest participant $p_i$ that is not corrupted, the value of each $\kappa_i$ must be the same.*
(2) *Confidentiality*
*Let a random symbolic protocol $\overline{\pi}_{random}$ be the same as the real symbolic protocol $\overline{\pi}$, except that a fresh random key $\kappa_{random}, (\kappa_{random} \in A)$ is output as the session key, instead of the real key $\kappa_{real}, (\kappa_{real} \in A)$. Then for any adversary strategy $\Psi$: $Pat(\Psi(\overline{\pi})) = Pat(\Psi(\overline{\pi}_{random}))_{[t_{random} \to t_{real}]}$ That is, the adversary cannot distinguish the real session key from a random key.*

(3) *Forward Secrecy*
*For each participant $p_i$ that is not corrupted, $[\text{“}erase\text{”}, s_i, p_i, \{p_1, p_2, \ldots, p_n\}\backslash p_i]$ must occur before $[\text{“}output\text{”}, Key, \kappa_i, s_i, p_i, \{p_1, p_2, \ldots, p_n\}\backslash p_i]$ in the trace. That is, the adversary cannot obtain the internal state of session $s_i$, after the session $s_i$ is end.*

## 4. Computational Protocols

In this section, we discuss computational protocols in UC security framework. Since computational protocol execution has been defined via PPT ITM, we only need to define the computational trace and computational security criterion of group key exchange in UC model.

### *4.1. Computational Trace*

Intuitively, computational trace is used to describe how an environment interacts with the honest participants running the computational protocol via the adversary. Formal definition of computational trace is as follow:

**Definition 12** (**Computational Trace**)**.** *Let $TRACE_{\pi,\mathcal{S},\mathcal{Z}}(k, z)$ be the computational trace of computational protocol $\pi$ with a dummy adversary $\mathcal{S}$, and an environment $\mathcal{Z}$ on security parameter $k$ and input $z$. The computational trace is the sequence of events $H_1, H_2, \ldots, H_n$, each of which is one of the triple:*

(1) *The initial event*
$[\text{“}initial\text{”}, sid, p_i, \{p_1, p_2, \ldots, p_n\}\backslash p_i]$ *written on the input tape of $p_i$ represents that $\mathcal{Z}$ activates the participant $p_i$ to establish a session sid of protocol $\pi$ with all participants that are in $\{p_1, p_2, \ldots, p_n\}\backslash p_i$.*
(2) *The adversary event*
$[\text{“}adversary\text{”}, m, p_i]$ *represents that the adversary $\mathcal{S}$ sends a message $m$ to the incoming communication tape of the participant $p_i$.*
(3) *The honest participant event*
$[\text{“}message\text{”}, m]$ *indicates that the participant $p_i$ sends a message $m$ to the incoming communication tape of the adversary $\mathcal{S}$.*
$[\text{“}erase\text{”}, er]$ *indicates that the participant $p_i$ erases its private key of session er from its work tape, and send er to the incoming communication tape the adversary $\mathcal{S}$.*
$[\text{“}output\text{”}, o]$ *indicates that the participant $p_i$ outputs o to its local output tape.*

Moreover, we denote $\{TRACE_{\pi,\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ as $EXEC_{\pi,\mathcal{S},\mathcal{Z}}$.

*4.2. Computational Security Criterion of Group Key Exchange*

Here we first define the functionality of bilinear pairings. It describes an ideal bilinear mapping operation for bilinear pairing. Formal definition of functionality $\mathcal{F}_{BP}$ is in Fig 2.

**Functionality $\mathcal{F}_{BP}$**

$\mathcal{F}_{BP}$ proceeds as follows, running with security parameter $k$, instance generator $IG$.

1. **Setup:** Upon receiving $(Setup, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$ from the participant $p_i$ for the first time, verify there is no record $(Setup, s, \{p_1, \ldots, p_n\})$ for the participant $p_i \in \{p_1, \ldots, p_n\}$. If so, Record $(Setup, s, \{p_1, \ldots, p_n\})$, compute $(q, \mathbb{G}_1, \mathbb{G}_2, g_1, AK\_Gen_1, \ldots, AK\_Gen_l)$ from $IG(k)$. Then compute $(K_{pri}, K_{pub})$ from $(q, \mathbb{G}_1, g_1)$. Record $(Pri, s, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i, K_{pri})$ and $(Pub, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, K_{pub})$, and generate a public delayed output of the latter to $p_i$.
2. **AK Generation:** Upon receiving a value $(AK\_Gen_i, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, m)$ from the participant $p_i$, compute $K_{ak}^i$ from $(q, \mathbb{G}_1, \mathbb{G}_2, g_1, AK\_Gen_i, K_{pri}, m)$, and generate a public delayed output $(AK_i, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, K_{ak}^i)$ to $p_i$.
3. **BP Generation:** Upon receiving a value $(BP\_Gen, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, m)$ from the participant $p_i$, do: If $p_i$ is corrupted, send $(BP\_Generate, s, \{p_1, p_2, \ldots, p_n\})$ to the adversary. Upon receiving $(BP, s, \{p_1, \ldots, p_n\}, \kappa)$ from the adversary, send it to $p_i$. Else if there is a record $(BP, s, \{p_1, \ldots, p_n\}, K_{bp})$ for $p_i \in \{p_1, \ldots, p_n\}$, generate a private delayed output of it to $p_i$. Else choose $K_{bp}$ from $\mathbb{Z}_q^*$ uniformly at a random, record $(BP, s, \{p_1, \ldots, p_n\}, K_{bp})$ and generate a private delayed output of it to $p_i$.
4. **Expiration:** Upon receiving a value $(Expire, s, p_i, \{p_1, p_2, \ldots, p_n\} \backslash p_i)$ from adversary, do nothing.
5. **Corruption:** Upon receiving a value $(Corrupt, p_i)$ from the adversary, mark $p_i$ as corrupted. Send all the records $(BP, s, \{p_1, p_2, \ldots, p_n\}, K_{bp})$ that has not been sent to $p_i$ and $(Pri, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, K_{pri})$ to the adversary.

Fig 2. The Functionality of Bilinear Pairing

Next we discuss security of bilinear pairings. Although different group key exchange protocols use bilinear pairings in different ways, their security is the same as Theorem 1.

**Theorem 1** (**Security of Group Key Exchange Protocols using Bilinear Pairings**). *The group key exchange protocols that use bilinear pairings securely realizes $\mathcal{F}_{BP}$, if the instance generator $IG$ satisfies BDDH assumption.*

*Proof.* The proof is by reduction. We show if there exist a PPT distinguisher $\mathcal{D}$ that distinguishes $(q, g_1^x, g_1^y, g_1^z, g_2^{xyz})$ from $(q, g_1^x, g_1^y, g_1^z, g_2^r)$, where $x, y, z, r \leftarrow \mathbb{Z}_q^*$, with non-negligible probability, then there exists a PPT environment $\mathcal{Z}$ that can distinguish its interaction with real protocols and real adversary $\mathcal{A}$ from ideal functionality $\mathcal{F}_{BP}$ and ideal $\mathcal{S}$. $\mathcal{D}$ runs $\mathcal{Z}$ as follow:
**Distinguisher $\mathcal{D}$**:
$\mathcal{D}$ is given $(g_1^x, g_1^y, g_1^z, g_2^c)$ and $(IG, k)$ as the input. It chooses $u, v, w$ from $\mathbb{Z}_{n+1}^*$ uniformly at random. Then it generates simulated participants $\{p_1, \dots, p_n\}$, chooses $i \leftarrow \{1, \dots, l(k)\}$ uniformly at random, and runs $\mathcal{Z}$ as follow:

(1) When $\mathcal{Z}$ activates the participant $p_j$ by $(Setup, s, p_j, \{p_1, \dots, p_n\} \backslash p_j)$, $\mathcal{D}$ activates its simulated participant $p_j$ with the same input. Compute $(q, \mathbb{G}_1, \mathbb{G}_2, g_1, e)$ from $IG(k)$.

(2) When $\mathcal{Z}$ activates the participant $p_j$ by $(AK\_Gen_j, s, p_j, \{p_1, \dots, p_n\} \backslash p_j, m)$, if $s = i$, each $r_i, (i \in \{1, \dots, n\})$ is chosen randomly from $\mathbb{Z}_q^*$, except that we choose $r_u, r_v, r_w$ randomly, and replace $g_1^{r_u}, g_1^{r_v}, g_1^{r_w}$ by $g_1^x, g_1^y, g_1^z$ respectively. Otherwise, $r_i, (i \in \{1, \dots, n\})$ is chosen randomly from $\mathbb{Z}_q^*$. Then $\mathcal{D}$ sends $(AK_j, s, p_j, \{p_1, \dots, p_n\} \backslash p_j, ak)$ to $\mathcal{Z}$. Here although different protocols have different $ak$, $ak$ always can be computed by $r_i, (i \in \{1, \dots, n\})$ and $g_1^{r_u}, g_1^{r_v}, g_1^{r_w}$.

(3) When $\mathcal{Z}$ activates the participant $p_i$ by $(BP\_Gen, s, p_i, \{p_1, \dots, p_n\} \backslash p_i, m)$, if $p_i$ is corrupted and $s = i$, halt and output a random number from $\{0, 1\}$. Else compute and send $(BP, s, \{p_1, \dots, p_n\}, K_{bp})$ to $\mathcal{Z}$ by $(pri, m, q, g_2)$. In particular, there is only one message that $\mathcal{D}$ cannot know. That is, if $s = i$, $g_2^{r_u r_v r_w}$ cannot be computed, since $\mathcal{D}$ do not know all the $r_u, r_v, r_w$. If necessary, $\mathcal{D}$ replace it by $g_2^c$.

(4) When $\mathcal{Z}$ activates the adversary $\mathcal{A}$ by $(Expire, s, p_i, \{p_1, \dots, p_n\} \backslash p_i)$, do nothing.

(5) When $\mathcal{Z}$ activates the adversary $\mathcal{A}$ by $(Corrupt, p_i)$, if $(BP, i, \{p_1, p_2, \dots, p_n\}, K_{bp})$ has not been sent to $\mathcal{Z}$ yet, halt and output a random num-

ber from $\{0,1\}$. Else send all the records $(BP, s, \{p_1, p_2, \ldots, p_n\}, K_{bp})$ and $(Pri, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, K_{pri})$ to the adversary.

(6) When $\mathcal{Z}$ outputs a bit $b$, halt and output $b$.

Here $l(k)$ is the upper bound of sessions that can be created in PPT. From the view of $\mathcal{Z}$ that runs as a sub-routine by $\mathcal{D}$, if $c = xyz$, it is distributed identically to the view of $\mathcal{Z}$ that interacts with a real bilinear pairing and a real adversary $\mathcal{A}$. Else if $c = r$, it is distributed identically to the view of $\mathcal{Z}$ that interacts with $\mathcal{F}_{BP}$ and $\mathcal{S}$. Therefore, assume that $\varepsilon(k)$ is the advantage that $\mathcal{Z}$ can distinguish its interaction with the real bilinear pairing and $\mathcal{A}$ from $\mathcal{F}_{BP}$ and $\mathcal{S}$, we have: $Adv_{\mathcal{D},IG}^{n-BDDH}(k) = \varepsilon(k)/l(k)$

This completes the proof. □

Finally the ideal functionality of group key exchange $\mathcal{F}_{GKE}$ is in Fig 3.

**Functionality $\mathcal{F}_{GKE}$**

$\mathcal{F}_{GKE}$ proceeds as follows, running with security parameter $k$ and an adversary $\mathcal{A}$, with participants $p_1, \ldots, p_n$.

1. **Session Establishment:** Upon receiving a value $(Establish, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$ from the participant $p_i$ for the first time, record and send it to $\mathcal{A}$.
2. **Key Generation:** Upon receiving $(Key\_Gen, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$ from the participant $p_i$, if there are already $n-1$ recorded tuples $(Establish, s, p_j, \{p_1, \ldots, p_n\} \backslash p_j)$ for $p_i \in \{p_1, \ldots, p_n\} \backslash p_j$. check if there is a record $(Key, s, \{p_1, p_2, \ldots, p_n\}, \kappa)$. If so, generate a private delayed output of it to $p_i$. Else if all participants are not corrupted, record and generate a private delayed output $(Key, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i, \kappa)$ to $p_i$, where $\kappa$ is chosen from $\{0,1\}^k$ uniformly at random. Else send $(Deliver, s, \{p_1, p_2, \ldots, p_n\})$ to the adversary. Upon receiving $(Key, s, \{p_1, \ldots, p_n\}, \kappa)$ from the adversary, record and generate a private delayed output of it to $p_i$.
3. **Expiration:** Upon receiving a value $(Expire, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$ from $\mathcal{A}$, do nothing.
4. **Corruption:** Upon receiving a value $(Corrupt, p_i)$ from $\mathcal{A}$, mark $p_i$ as corrupted, if there is any record $(Key, s, \{p_1, \ldots, p_n\}, \kappa)$ that has not been sent to $p_i$, send it to the adversary.

Fig 3. The Functionality of Group Key Exchange

## 5. Simple Protocols

### *5.1. The Syntax of Simple Protocols*

Simple protocols are used to specify operations for message communications (sending and receiving) and manipulations (bilinear mapping and signature). They are written by a specific programming language that keeps each operation efficiently computed and restricts to commands that reflect the structure of symbolic model.

We define simple protocols as an extension of [11] and [12]. In comparison, we define calculus for bilinear pairings, and we relax the presentation to group key exchange protocols. Formal definition of simple protocols is as follow:

**Definition 13** (**Syntax of Simple Protocols**). *A simple protocol $\pi$ is a n-tuple of programs $\pi = (\pi_1, \ldots, \pi_n)$, each of which is given by the grammar in Fig 4. , where $vc, vc_0, vc_1, vc_2$ stand for constants and $v, v_0, v_1$ stand for variables.*

$PROGRAM ::= initialize(sid, pid_i, \{pid_1, \ldots, pid_n\} \backslash pid_i); COMMAND$
$COMMAND ::= COMMAND; COMMAND | done | receive(v) | send(vc) |$
$output(vc) | newrandom(v) | pair(vc_0, vc_1, v) | setup(vc, v) | ak\_gen_1(vc_1, vc_2, v) |$
$\ldots | ak\_gen_n(vc_1, vc_2, v) | bp\_gen(vc_0, vc_1, v) | erase(vc) | sep(vc, v_0, v_1) | sign(vc_0,$
$vc_1, v) | verify(vc_0, vc_1, vc_2, v)$

*Fig 4. Syntax of Simple Protocols*

### *5.2. Symbolic Semantics of Simple Protocols*

We show the formal definition for the symbolic semantics of simple protocols as follow:

**Definition 14** (**Symbolic Semantics of Simple Protocols**). *Let $\pi = (\pi_1, \ldots, \pi_n)$ be a simple protocol. Let $\overline{\pi}$ be the symbolic protocol of $\pi$ where the set of states $S$ consist of a program counter $\Gamma$ that indicates the next command to execute, and a store command $\Delta$ that maps variables in $\pi$ to the corresponding symbols in $A$. For all $(\Gamma, \Delta) \in S, m \in A, s \in SID, p \in PID$, the mapping $\overline{\pi}$ is defined on the commands in $\pi = (\pi_1, \ldots, \pi_n)$ as follow:*
*(1) If $\Gamma$ points to the command of send(vc), then $\overline{\pi}(s, p, (\Gamma, \Delta), m) \rightarrow$ (“message”$, \Delta(vc), (\Gamma', \Delta))$ , $\Gamma'$ points to the next command.*
*(2) If $\Gamma$ points to the command of output(vc), then $\overline{\pi}(s, p, (\Gamma, \Delta), m) \rightarrow$ (“output”$, vc), (\Gamma', \Delta))$, $\Gamma'$ points to the next command.*

*(3) If $\Gamma$ points to one of the following commands, then $\overline{\pi}(s, p, (\Gamma, \Delta), m) \rightarrow (s, p, (\Gamma', \Delta'), m)$, where $\Gamma'$ points to the next command and $\Delta'$ is equal to $\Delta$, except that:*
$receive(v) : \Delta'(v) = m$.
$erase(vc) : \Delta'(v) = \xi$
$newrandom(v) : \Delta'(v)$ *is the first element of* $R$ *that is not in the range of* $\Delta$.
$pair(vc_1, vc_2, v) : \Delta'(v) = Pair(\Delta(vc_1), \Delta(vc_1))$.
$setup(vc, v) : \Delta'(v) = Setup(\Delta(vc))$.
$ak\_gen_i(vc_1, vc_2, v) : \Delta'(v) = AK\_Gen_i(\Delta(vc_1), \Delta(vc_2))$.
$bp\_gen(vc_1, vc_2, v) :$ *If* $\Delta(vc_1) \in Pri$, $\Delta(vc_2) \in A$,
$\Delta'(v) = BP\_Gen(\Delta(vc_1), \Delta(vc_2))$. *Otherwise,* $\Delta'(v) = \varrho$.
$sep(vc, v_1, v_2) :$ *If* $\Delta(vc) = Pair(\Delta(vc_1), \Delta(vc_2))$,
$(\Delta'(v_1), \Delta'(v_2)) = Sep(\Delta(vc))$. *Otherwise,* $\Delta'(v) = \varrho$.
$sign(vc_1, vc_2, v) : \Delta'(v) = Sign(\Delta(vc_1), \Delta(vc_2))$.
$verify(vc_1, vc_2, vc_3, v) : \Delta'(v) = Verify(\Delta(vc_1), \Delta(vc_2), \Delta(vc_3))$.

## *5.3. Computational Semantics of Simple Protocols*

We now define the computational semantics of simple protocols as follow:

**Definition 15 (Computational Semantics of Simple Protocols).** *Let a PPT ITM $\pi = (\pi_1, \ldots, \pi_n)$ be a simple protocol. The state set $S_M$ consists of $\{$"initial"$\} \cup S_1 \cup \cdots \cup S_n$ , where "initial" represents the initial state of $M_\pi$, and each state $S_i = (\pi_i, \Delta_i, \Gamma_i)$,$(i \in \{1, \ldots, n\})$ represents the protocol of the corresponding participants $\pi_i$, a store command $\Delta_i$ which maps from variable names in $\pi_i$ to locations on the work tape, and a program counter $\Gamma_i$ which indicates the current command of $\pi_i$. To encode the execution of each $\pi_i$, the transition function is defined over $S_M$ as follow:*

(1) *If $M_\pi$ is in the initial state "initial", it will first read the security parameter $k$, a session identity $s_i$, a participant identity $p_i$, and a set of participant identities $\{p_1, \ldots, p_n\}\backslash p_i$. Then $M_\pi$ initializes the storage and writes $[$"initial"$, s_i, p_i, \{p_1, \ldots, p_n\}\backslash p_i]$ to indicate that $p_i$ is initialized with all the participants in $\{p_1, \ldots, p_n\}\backslash p_i$ in the session $s_i$. Finally, it sets the program counter $\Gamma_i$ to the next command and executes it.*
(2) *After initialization, the transition function continues to execute the command of $\pi_i$ by program counter $\Gamma_i$:*
*receive($v$): If the command has already been executed in this activation, $M_\pi$ waits to be reactivated. Otherwise, it first reads the message from its*

*incoming communication tape and stores it in $v$. Then it instructs $\Gamma_i$ to the next command and executes it.*
*send($vc$): $M_\pi$ writes $vc$ to the incoming communication tape of the adversary. Then it instructs $\Gamma_i$ to the next command and executes it.*
*output($vc$): $M_\pi$ writes $vc$ to its local output tape. Then it instructs $\Gamma_i$ to the next command and executes it.*
*newrandom($v$): $M_\pi$ first generates a random number $r \leftarrow \{0,1\}^k$, then it stores $[\text{“}random\text{”}, r]$ in $v$, and instructs $\Gamma_i$ to the next command and executes it.*
*pair($vc_1, vc_2, v$): $M_\pi$ first stores $[\text{“}pair\text{”}, vc_0, vc_1]$ in $v$, then it instructs $\Gamma_i$ to the next command and executes it.*
*setup($vc, v$): $M_\pi$ first sends $(Setup, vc)$ to $\mathcal{F}_{BP}$, where $vc = (s, p_i, \{p_1, \dots, p_n\} \backslash p_i)$. Then it reads $(Pri, s, p_i, vc, K_{pri})$ and $(Pub, s, vc, K_{pub})$ from its subroutine output tape. After that, it stores $[\text{“}setup\text{”}, vc, K_{pri}, K_{pub}]$ in $v$, Finally, it instructs $\Gamma_i$ to the next command and executes it.*
*ak_gen$_i$($vc_1, vc_2, v$): $M_\pi$ first sends $(AK\_Generate_i, vc_1, vc_2)$ to $\mathcal{F}_{BP}$, where $vc_1 = (s, p_i, \{p_1, \dots, p_n\} \backslash p_i)$. Then it reads $(AK_i, vc_1, K_{ak})$ from its subroutine output tape. After that, it stores $[\text{“}ak_i\text{”}, vc_1, vc_2, K^i_{ak}]$ in $v$. Finally it instructs $\Gamma_i$ to the next command and executes it.*
*bp_gen($vc_1, vc_2, v$): $M_\pi$ first sends $(BP\_Generate, vc_1, vc_2)$ to $\mathcal{F}_{BP}$, where $vc_1 = (s, p_i, \{p_1, \dots, p_n\} \backslash p_i)$. Then it waits until $(BP, vc_1, K_{bp})$ is written to its subroutine output tape. After that, it stores $[\text{“}bp\text{”}, vc_1, vc_2, K_{bp}]$ in $v$. Finally, it instructs $\Gamma_i$ to the next command and executes it.*
*erase($vc$): $M_\pi$ deletes the value of $vc$ from its work tape, instructs $\Gamma_i$ to the next command and executes it.*
*sep($vc, v_1, v_2$): If the value of $vc$ is $[\text{“}pair\text{”}, a, b]$, $M_\pi$ stores $vc_1 = a$ and $vc_2 = b$, then it instructs $\Gamma_i$ to the next command and executes it.*
*sign($vc_1, vc_2, v$): $M_\pi$ first sends $(Sign, vc_1, vc_2)$ to $\mathcal{F}_{CERT}$, where $vc_1 = (s, p_i, \{p_1, \dots, p_n\} \backslash p_i)$, and $vc_2$ is the message to be signed. It waits until $\mathcal{F}_{CERT}$ writes the signature $\sigma$ on the subroutine output tape. Then it stores $[\text{“}sign\text{”}, vc_1, vc_2, \sigma]$ in $v$. Finally, it instructs $\Gamma_i$ to the next command and executes it.*
*verify($vc_1, vc_2, vc_3, v$): $M_\pi$ first sends $(Verify, vc_1, vc_2, vc_3)$ to $\mathcal{F}_{CERT}$, where $vc_1 = (s, p_i, \{p_1, \dots, p_n\} \backslash p_i)$, $vc_2$ is the messages, and $\sigma$ is the last element of $v_3$, where $v_3 = [\text{“}sign\text{”}, s, m', \sigma']$. It waits until $\mathcal{F}_{CERT}$ writes the bit $b$ on the subroutine output tape. It outputs $[\text{“}verify\text{”}, vc_1, vc_2, vc_3, b]$. If $b = 1$, it instructs $\Gamma_i$ to the next command and executes it. Otherwise, it halts.*

## 6. Mapping Algorithm and Mapping Lemma

In this section, we discuss how to map computational trace to symbolic trace. Furthermore, we prove that the mapping algorithm is always valid except with negligible probability. It is used to bridge the gap between symbolic model and computational model. We start with the definition of mapping algorithm.

**Definition 16** (**Mapping Algorithm**). *Let $\pi$ be a simple protocol and $TRACE_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ be the computational trace of $\pi$ with security parameter $k$, environment $\mathcal{Z}$, adversary $\mathcal{A}$ and input $z$. Let $\overline{\pi}$ be the corresponding symbolic protocol and $\overline{t}$ be the symbolic trace of $\overline{\pi}$ against symbolic adversary $\overline{\mathcal{A}}$. We define a mapping algorithm $\delta$ from $TRACE_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ to $\overline{t}$ by two steps as follow:*
*In the first step, the algorithm reads all the computational trace $TRACE_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ character by character, and maps them from bit-strings to the corresponding elements in the symbolic algebra $A$ according to the cases below:*
*Map $[\text{“}sid\text{”}, s]$ to $\boldsymbol{s}$, where $\boldsymbol{s}$ is the first element of $SID$ in the range of $\delta$.*
*Map $[\text{“}pid\text{”}, p]$ to $\boldsymbol{p}$, where $\boldsymbol{p}$ is the first element of $PID$ in the range of $\delta$.*
*Map $[\text{“}pri\text{”}, vc]$ to $\boldsymbol{pri}$, where $\boldsymbol{pri}$ is the first element of $Pri$ in the range of $\delta$.*
*Map $[\text{“}pub\text{”}, vc]$ to $\boldsymbol{pub}$, where $\boldsymbol{pub}$ is the first element of $Pub$ in the range of $\delta$.*
*Map $[\text{“}sk\text{”}, vc]$ to $\boldsymbol{sk}$, where $\boldsymbol{sk}$ is the first element of $SK$ in the range of $\delta$.*
*Map $[\text{“}vk\text{”}, vc]$ to $\boldsymbol{vk}$, where $\boldsymbol{vk}$ is the first element of $VK$ in the range of $\delta$.*
*Map $[\text{“}setup\text{”}, vc, K_{pri}, K_{pub}]$ to $(\boldsymbol{sk}, \boldsymbol{pk}) = Setup()$, where $\boldsymbol{sk}$ is the first element of $Pri$ and $\boldsymbol{pk}$ is the first element of $Pub$ in the range of $\delta$.*
*Map $[\text{“}random\text{”}, r]$ to $\boldsymbol{r}$, where $\boldsymbol{r}$ is the first element of $R$ in the range of $\delta$, if $\delta([\text{“}random\text{”}, r])$ has not been defined yet.*
*Map $[\text{“}pair\text{”}, a, b]$ to $Pair(\delta(a), \delta(b))$, if $\delta(a)$ or $\delta(b)$ has already been defined. Else, map it to $\varrho$.*
*Map $[\text{“}ak_i\text{”}, vc_1, vc_2, K^i_{ak}]$ to $AK\_Gen_i(\boldsymbol{pri}, \delta(vc_2))$, where $\boldsymbol{pri} = \delta(\text{“}pri\text{”}, vc_1)$.*
*Map $[\text{“}bp\text{”}, vc_1, vc_2, K_{bp}]$ to $BP\_Gen(\boldsymbol{pri}, \delta(vc_2))$, where $\boldsymbol{pri} = \delta(\text{“}pri\text{”}, vc_1)$.*
*Map $[\text{“}sign\text{”}, vc_1, vc_2, \sigma]$ to $Sig(\boldsymbol{pri}, \delta(vc_2))$, where $\boldsymbol{sk} = \delta(\text{“}sk\text{”}, vc_1)$.*
*Map $[\text{“}verify\text{”}, vc_1, vc_2, vc_3, b]$ to $VerSig(\boldsymbol{vk}, \delta(m), \delta(\sigma))$, where $\boldsymbol{vk} = \delta(\text{“}vk\text{”}, vc_1)$.*
*Map garbage string to $\varrho$, empty string to $\xi$, continue symbol to $\top$, termination symbol to $\bot$*

*Map "Establish" and "Key" to the same symbols.*
*Map "corrupt" and "expire" to the same symbols.*
*In the second step, the algorithm constructs the symbolic trace as follow:*

*(1) When translating* $[\text{“}initial\text{”}, s, p_i, \{p_1, \ldots, p_n\}\backslash p_i]$ *to the symbolic event, map it to* $[\text{“}input\text{”}, \boldsymbol{s}, \boldsymbol{p_i}, \{\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}\}\backslash \boldsymbol{p_i}, S_{i,init}]$*, where* $\boldsymbol{s} = \delta(\text{“}sid\text{”}, s)$*,* $\boldsymbol{p_i} = \delta(\text{“}pid\text{”}, p_i)$*,* $\boldsymbol{p_j} = \delta(\text{“}pid\text{”}, p_j), (\boldsymbol{j} \in [1, n]\backslash \boldsymbol{i})$*.*

*(2) When translating* $[\text{“}adversary\text{”}, m, p_i]$ *to the symbolic event, if m can be generated by the adversary from previous messages of the trace, map it to* $[\text{“}deliver\text{”}, m, p_i]$*. Otherwise, map it to* $[\text{“}fail\text{”}, m]$*.*

*(3) When translating* $[\text{“}message\text{”}, m]$*,* $[\text{“}erase\text{”}, er]$*, or* $[\text{“}output\text{”}, o]$ *to the symbolic event, map them to the same symbols.*

Next we show that the mapping algorithm is valid. This is done by proving mapping lemma as follow:

**Lemma 1 (Mapping Lemma).** *Assume that* $\delta$ *is the mapping algorithm that maps from computational trace* $TRACE_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ *to symbolic trace* $\bar{t}$ *as in Definition 16. For all simple protocols* $\pi$*, environment* $\mathcal{Z}$*, adversary* $\mathcal{A}$*, input z and security parameter k:* $Pr[t \leftarrow EXEC_{\pi,\mathcal{A},\mathcal{Z}} : \bar{t}$ *is not a valid symbolic trace for* $\overline{\pi}] \leq neg(k)$*.*

*Proof.* We first construct a parse tree of messages, Next we show that $\bar{t}$ contains $[\text{“}fail\text{”}, m]$ with negligible probability. Finally, we show that $\bar{t}$ is always valid if it does not contain $[\text{“}fail\text{”}, m]$.

The parse tree of a message $m$ is the tree whose root node is $m$, leaf nodes are atomic messages in $A$, and edges from node $m_1$ to node $m_2$ indicates that $m_1$ can be derived from $m_2$ by adversary.

Let $C^j_{Adv}, (j < i)$ be an adversarial closure that can be generated in the trace prior to $m_i$. If all children nodes are in $C^j_{Adv}$, the parent node is in $C^{i-1}_{Adv}$. As a result, if $\bar{t}$ contains $[\text{“}fail\text{”}, m_i]$, it represents that $m_i \notin C^{i-1}_{Adv}$, where $m_i \in H_i$. That is, the symbolic adversary cannot derive $m_i$ from previous messages of the symbolic trace, while the computational adversary can derive it. Therefore, there must be a leaf node $m_l$ and a node $m_*$ in the path from $m_i$ to $m_l$, such that all the nodes from $m_*$ to $m_l$ are not in $C^j_{Adv}$.

The environment $\mathcal{Z}$ can instruct an adversary $\mathcal{A}_1$ to generate a bit-string $\mathtt{m_i}$, such that $\delta(\mathtt{m_i}) = m_i$. Then it instructs another adversary $\mathcal{A}_2$ to generate a bit-string $\mathtt{m_*}$, such that $\delta(\mathtt{m_*}) = m_*$. where $\mathtt{m_*}$ is the message on the path from $m_i$ to $m_l$. Finally, it maps the bit-strings $\mathtt{m_i}$ to symbols using $\delta$

recursively. That is, it parses from $m_i$ to $m_l$ recursively. There are three possibilities:

(1) $m_*$ is in the form of $BP\_Gen(pri, ak)$. Since the participants use $\mathcal{F}_{BP}$ to generate the secret key $m_{bp}$, if $m_*$ is not in the closure $C^j_{Adv}$, then $\mathcal{A}_2$ has to use $\mathcal{F}_{BP}$ to generate it. Since $\mathcal{F}_{BP}$ generates it from $\{0,1\}^k$ uniformly at random, the probability that $\mathcal{F}_{BP}$ generates two same bilinear paring keys is negligible. Otherwise, $m_*$ is in the form of $[\text{“}corrupt\text{”}, p_i]$ which must be in $C^j_{Adv}$.

(2) $m_*$ is in the form of $r$. Since $r$ is chosen from $\{0,1\}^k$ uniformly at random, the probability that $\mathcal{A}_2$ guesses right is negligible.

(3) $m_*$ is in the form of $Sign(sk, m)$. The participant uses $\mathcal{F}_{CERT}$ to generate signatures. Since $m_*$ is not in the closure $C^j_{Adv}$, $\mathcal{A}_2$ must create a valid signature for the message, which is not generated by the honest participant. Since $\mathcal{F}_{CERT}$ provides an ideal functionality of digital signature, the probability that $\mathcal{A}_2$ succeeds is negligible.

Therefore, the probability of which the event $[\text{“}fail\text{”}, m]$ occurs is negligible. Furthermore, If $\bar{t}$ does not contain the event of the form $[\text{“}fail\text{”}, m]$, it is valid by the Definition 8. In addition, the mapping algorithm is valid for both the initial event and honest participant event based on the Definition 14.

This completes the proof. □

After that, we prove the computational soundness of the symbolic security criterion as follow:

**Theorem 2 (Computational Soundness of Symbolic Security Criterion).** *For any adversary strategy $\Psi$, Let a simple protocol $\pi$ that use bilinear pairings exchange and secure digital signatures securely realize $\mathcal{F}_{BP}$. If $\pi$ has a mapping $\delta$ from computational trace $TRACE_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ to symbolic trace, and $\overline{\pi}$ is a SGKE protocol, then $\pi$ securely realizes $\mathcal{F}_{GKE}$.*

*Proof.* We need to prove that $\overline{\pi}$ is not a SGKE protocol, if the environment $\mathcal{Z}$ can distinguish its interaction with the real protocol $\pi$ and the real adversary $\mathcal{A}$, from its interaction with the ideal functionality $F_{GKE}$ and the ideal adversary $\mathcal{S}$.

First of all, we need to construct a simulator (i.e., an adversary) to show how $\mathcal{F}_{GKE}$ and $\mathcal{S}$ simulate $\pi$ and $\mathcal{A}$. The simulator proceeds as follow:

(1) The simulator simulates each participant $p_i$, but none of these simulated participants are running.

(2) When the simulator receives a message $(Establish, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$ from $\mathcal{F}_{GKE}$, which indicates that the participant $p_i$ has already been initialized by the environment, pass it to $\mathcal{F}_{GKE}$. The simulator activates the simulated participant $p'_i$ by the same input.

(3) When the simulator receives a message from the environment to the participant $p_i$, forward it on the incoming communication tape of the simulated participant $p'_i$.

(4) When $p'_i$ sends a message on its incoming communication tape, it sends the message to the environment.

(5) When $p'_i$ generate an output $(Key, \kappa, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$, forward it to $\mathcal{F}_{GKE}$.

(6) When the simulator receives a message $(Expire, s, p_i, \{p_1, \ldots, p_n\} \backslash p_i)$ from the environment, forward it to $\mathcal{F}_{GKE}$.

(7) When the simulator receives a message $(Corrupt, p_i)$ from the environment, forward it to $\mathcal{F}_{GKE}$.

If all honest participants are not marked "*corrupted*", and the environment interacts with $\mathcal{S}$ and $\mathcal{F}_{GKE}$, the session key is a random number. Therefore, if the environment can distinguish whether it interacts with the real protocol $\pi$ and the real adversary $\mathcal{A}$ from the ideal functionality $F_{GKE}$ and the dummy adversary $\mathcal{S}$, then $\mathcal{A}$ must distinguish the real session key from a random key, since $\pi$ securely realizes $\mathcal{F}_{GKE}$ and digital signatures are secure. That is, $\overline{\pi}$ is not a SGKE protocol.

This completes the proof. $\square$

Note that Theorem 2 implies the group key exchange protocol using bilinear pairings securely realizes $\mathcal{F}_{BP}$, and the digital signature securely realizes $\mathcal{F}_{CERT}$, according to the definition 15. In addition, the universally composable theorem with joint state in [10] makes us only need to analyze a single session of $\pi$. Therefore, we only need to analyze a single session of $\overline{\pi}$ in symbolic model.

## 7. Security Reduction

In this section, we reduce the security of a group key exchange session that use bilinear pairings exchange with arbitrary number of participants to the same session with three participants in symbolic model.

**Theorem 3.** *Let $\overline{\pi}$ be a symbolic simple session that uses bilinear pairings exchange with $n$ participants, where $(n \in \mathbb{N}, n > 3)$, and $\overline{\pi}_3$ be the same* *session with three participants. For any adversary strategy $\Psi$, if $\overline{\pi}_3$ is a SGKE session, then $\overline{\pi}$ is a SGKE session.*

*Proof.* First, we limit the adversary is passive. Assume that $\overline{\pi}_i$ is the session with $i$ participants, $(q, \mathbb{G}_1, \mathbb{G}_2, g_1, e) \leftarrow IG(k)$ be the parameters of the bilinear pairings used in $\overline{\pi}_i$. We prove this theorem by mathematical induction. Since $\overline{\pi}_3$ is a SGKE session, we only need to prove that if $\overline{\pi}_k$ is a SGKE session, then $\overline{\pi}_{k+1}$ is also a SGKE session. Therefore, for $\overline{\pi}_k$, we construct a simulation for $\overline{\pi}_{k+1}$ as follow:

(1) $\overline{\mathcal{A}}_k$ executes $\overline{\pi}_k$ and obtains all the messages include all the public keys $g_1^{r_1}, \ldots, g_1^{r_k}$, $AK_1^k, \ldots, AK_n^k$ and an output $K_c^k$.

(2) $\overline{\mathcal{A}}_k$ simulates all the honest participants $p_1, \ldots, p_k, p_{k+1}$ in $\overline{\pi}_{k+1}$. In particular, it generates the private key, public key of the participant $p_{k+1}$ by itself. Therefore, it can compute all the messages $AK_1^{k+1}, \ldots, AK_{k+1}^{k+1}$ by the method of undetermined coefficients based on the private key of $p_{k+1}$, $AK_1^k, \ldots, AK_k^k$ and $K_c^k$.

(3) If $\overline{\mathcal{A}}_{k+1}$ activates a participant $p_i$, it chooses the corresponding message from the public key $g_1^{r_i}$ and $AK_1^{k+1}, \ldots, AK_{n+1}^{k+1}$.

Note that from the point of $\overline{\mathcal{A}}_{k+1}$, if $K_c^k$ is the real session key, the interaction with this simulation is the same as the interaction with the real symbolic session. The probability that $\overline{\mathcal{A}}_k$ considers $K_c^k$ as the real session key of $\overline{\pi}_k$, equals the probability that $\overline{\mathcal{A}}_{k+1}$ considers $K_c^{k+1}$ as the real session key of $\overline{\pi}_{k+1}$. Otherwise, if $K_c^k$ is a random element of key space, the interaction with this simulation is the same as the interaction with the random symbolic session, except that part of the assistant keys are random numbers. If $\overline{\mathcal{A}}_{k+1}$ can distinguish those assistant keys are random numbers, the probability that $\overline{\mathcal{A}}_k$ considers $K_c^k$ as a random number in the execution of $(\overline{\pi}_k)_{random}$, equals 1. Otherwise, the probability that $\overline{\mathcal{A}}_k$ considers $K_c^k$ as a random number in the execution of $(\overline{\pi}_k)_{random}$, equals the probability that $\overline{\mathcal{A}}_{k+1}$ considers $K_c^{k+1}$ as a random number in the execution of $(\overline{\pi}_{k+1})_{random}$. Therefore, the probability that $\overline{\mathcal{A}}_k$ considers $K_c^k$ as a random number in the execution of $(\overline{\pi}_k)_{random}$ is not less than the probability that $\overline{\mathcal{A}}_{k+1}$ considers $K_c^{k+1}$ as a random number in the execution of $(\overline{\pi}_{k+1})_{random}$.

Above all, if there is a symbolic trace $\overline{t}_{k+1}$, such that $\overline{\mathcal{A}}_{k+1}$ is not a SGKE session, then $\overline{\mathcal{A}}_k$ can compute the corresponding symbolic trace $\overline{t}_k$ by $\overline{t}_{k+1}$, such that $\overline{\mathcal{A}}_k$ is not a SGKE session, since it knows the private key and signature key of $p_{k+1}$.

Furthermore, it can be used for sessions in the active settings. More specifically, it works even if the session uses digital signatures. In that case, we assume that each participant $p_i, (i \in [1,k])$ has a sign oracle to return the signature of all the messages input by $\overline{\mathcal{A}}_k$. Therefore, it can send arbitrary messages it needs to $p_i$. The proof is still valid then. As a result, if the digital signatures is secure, the conclusion of Theorem 3 is still valid.

This completes the proof. $\square$

**Example:** We show how $\overline{\mathcal{A}}_k$ simulates $(\alpha, \beta, \gamma)$-BBD session among $k+1$ participants via the same session with $k$ participants. Each session of $\overline{\pi}_{k+1}$ is simulated by $\overline{\pi}_k$ as follow:

(1) $\overline{\mathcal{A}}_k$ runs $(\alpha, \beta, \gamma)$-BBD session among $k$ participants to obtain all the public keys $g_1^{r_1}, \ldots, g_1^{r_k}$, assistant keys $g_2^{\alpha r_{k-2} r_{k-1} r_1 + \beta r_{k-1} r_1 r_2 + \gamma r_1 r_2 r_3}, \ldots,$ $g_2^{\alpha r_{k-2} r_{k-1} r_k + \beta r_{k-1} r_k r_1 + \gamma r_k r_1 r_2}$, and an output $K_c^k$ that is $g_2^{r_1 r_2 r_3 + \cdots + r_{k-1} r_k r_1 + r_k r_1 r_2}$ or a random element of $\mathbb{G}_2$.

(2) $\overline{\mathcal{A}}_k$ generates a private key $r_{k+1}$ and a public key $g^{r_{k+1}}$ for the $(k+1)^{th}$ participant.

(3) $\overline{\mathcal{A}}_k$ computes $g_2^{r_1 r_2 r_3}, \ldots, g_2^{r_{k-1} r_k r_1}, g_2^{r_k r_1 r_2}$ by applying undetermined coefficient to all the assistant keys and output obtained in (1).

(4) $\overline{\mathcal{A}}_k$ computes all the public keys $g_1^{r_1}, \ldots, g_1^{r_k}, g^{r_{k+1}}$, assistant keys $g_2^{\alpha r_{k-1} r_k r_1 + \beta r_k r_1 r_2 + \gamma r_1 r_2 r_3}, \ldots, g_2^{\alpha r_{k-1} r_k r_{k+1} + \beta r_k r_{k+1} r_1 + \gamma r_{k+1} r_1 r_2}$, and an output $K_c^{k+1}$ that is $g_2^{r_1 r_2 r_3 + \cdots + r_k r_{k+1} r_1 + r_{k+1} r_1 r_2}$ or a random element of $\mathbb{G}_2$, based on $g_2^{r_1 r_2 r_3}$, $\ldots, g_2^{r_{k-1} r_k r_1}, g_2^{r_k r_1 r_2}$ and $r_{k+1}$.

(5) $\overline{\mathcal{A}}_k$ run $\overline{\mathcal{A}}_{k+1}$ to simulates $(\alpha, \beta, \gamma)$-BBD session among $k+1$ participants by all the messages in (4).

Note that from the point of $\overline{\mathcal{A}}_{k+1}$, if $K_c^k$ is the real session key, it interacts with this simulation is the same as it interacts with the real symbolic session. Otherwise, if $K_c^k$ is a random number, it interacts with this simulation is the same as it interacts with the random symbolic session, except that part of assistant keys are random numbers.

Let $\overline{\mathcal{A}}_{k+1}$ output 0, if it considers $K_c^{k+1}$ as $g_2^{r_1 r_2 r_3 + \cdots + r_k r_{k+1} r_1 + r_{k+1} r_1 r_2}$. Otherwise, let $\overline{\mathcal{A}}_{k+1}$ output 1, if it considers $K_c^{k+1}$ as a random element of $\mathbb{G}_2$. Suppose the probability that $\overline{\mathcal{A}}_{k+1}$ can distinguish $g_2^{r_1 r_2 r_3 + \cdots + r_k r_{k+1} r_1 + r_{k+1} r_1 r_2}$ from a random element of $\mathbb{G}_2$ is denoted as $\varepsilon(k)$, we have:

$$\begin{aligned}\varepsilon(k) = {} & Pr[K_c^{k+1} \text{ is the real session key}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 0] \\ & + Pr[K_c^{k+1} \text{ is a random number}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 1]\end{aligned}$$

Assume that $\alpha$ is the probability that $\overline{\mathcal{A}}_{k+1}$ can distinguish if the assistant keys are random numbers or not, and $\varepsilon'(k)$ is the probability that $\overline{\mathcal{A}}_k$ can distinguish ${g_2}^{r_1r_2r_3+\cdots+r_{k-1}r_kr_1+r_kr_1r_2}$ from a random element of $\mathbb{G}_2$, we have:

$$\begin{aligned}
\varepsilon'(k) &= Pr[K_c^k \text{ is the real session key}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 0] \\
&\quad + Pr[K_c^k \text{ is a random number}] * (Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 1] * (1-\alpha) + 1 * \alpha) \\
&\geq Pr[K_c^k \text{ is the real session key}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 0] \\
&\quad + Pr[K_c^k \text{ is a random number}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 1] * ((1-\alpha) + \alpha) \\
&= Pr[K_c^k \text{ is the real session key}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 0] \\
&\quad + Pr[K_c^k \text{ is a random number}] * Pr[\overline{\mathcal{A}}_{k+1} \text{ outputs } 1] \\
&= \varepsilon(k)
\end{aligned}$$

Therefore, if there is a symbolic trace $\bar{t}_{k+1}$, such that $\overline{\mathcal{A}}_{k+1}$ is not a SGKE session, then $\overline{\mathcal{A}}_k$ can compute the corresponding symbolic trace $\bar{t}_k$ by $\bar{t}_{k+1}$, such that $\overline{\mathcal{A}}_k$ is not a SGKE session.

## 8. Automate Analysis Simple Protocols by ProVerif

In this section, we design two group key exchange protocols based on $(\alpha, \beta, \gamma)$-BBD protocol, and take them as examples to illustrate how to analyze group key exchange protocols that use bilinear pairings exchange and digital signatures based on our approach.

Theorem 2 proves if a simple protocol of group key exchange is a SGKE protocol, then it securely realizes $\mathcal{F}_{GKE}$. Applying the universally composable theorem with joint state in [10], we only need to analyze a single group key exchange session to resist insider attack in symbolic model. Furthermore, there is a compiler to translate secure group key exchange protocols to resist outsider attack to secure protocols to resist insider attack automatically [21]. Therefore, we only need to design a group key exchange protocol to resist outsider attack.

We start to apply the compiler in the work of [22] to design a group key exchange protocol based on $(\alpha, \beta, \gamma)$-BBD protocol as follow:
**Protocol Description of Compiled $(\alpha, \beta, \gamma)$-BBD Protocol.** *Assume that $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic group of the same prime order $q$ with respective generators $g_1$ and $g_2$. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ such that $e(g_1, g_1) = g_2$. Assume that each participant has already generated a signature key $sk_i$ and the corresponding verification key $vk_i$. In addition, each participants has known the verification keys of others before executing the protocol, all the indexes are taken modulo $n$, and $i \in [0, n-1]$. The protocol proceeds among*

*all participants $\{p_i\}$ as follow:*
(1) *Each $p_i$ chooses a random number $x_i \in \mathbb{Z}_q^*$ and broadcasts it to all the others.*
(2) *Each $p_i$ chooses a random number $r_i \in \mathbb{Z}_q^*$, keeps $r_i$ secret, computes $Z_i = g_1^{r_i}$, and broadcasts $S_i = (Z_i, Sig_{sk_i}(Z_i, (p_0, x_0, p_1, x_1, p_2, x_2)))$*
(3) *Each $p_i$ verifies the messages $Verify(S_j, vk_j)$ that it has received. If all of them are valid, $p_i$ computes: $X_i = e((Z_{i-2}, Z_{i-1})^{\alpha r_i}) \cdot e((Z_{i-1}, Z_{i+1})^{\beta r_i}) \cdot e((Z_{i+1}, Z_{i+2})^{\gamma r_i}) = g_2^{\alpha r_{i-2} r_{i-1} r_i + \beta r_{i-1} r_i r_{i+1} + \gamma r_i r_{i+1} r_{i+2}}$, , and broadcasts $T_i = (X_i, Sig_{sk_i}(X_i, (p_0, x_0, p_1, x_1, p_2, x_2)))$.*
(4) *Each $p_i$ verifies the messages $Verify(T_j, vk_j)$ that it has received. If all of them are valid, $p_i$ computes the session key: $K = g_2^{\sum_{i=0}^{n-1} r_i r_{i+1} r_{i+2}}$.*

Next we prove that the compiled $(\alpha, \beta, \gamma)$-BBD protocol is not a SGKE protocol.

**Theorem 4.** *The compiled $(\alpha, \beta, \gamma)$-BBD protocol is not a SGKE protocol.*

We analyze compiled protocol by ProVerif. Since the instance generator of the bilinear pairings used in the $(\alpha, \beta, \gamma)$-BBD protocol has been proved to satisfy BDDH assumption in the work of [23], it securely realizes $\mathcal{F}_{BP}$ based on Theorem 1. As a result, we only need to analyze a compiled session with three participants based on Theorem 2 and Theorem 3.

The specific program is shown in Fig 5 of Appendix. Note that the function *Setup* is implemented in two steps: (1) generate a random number as the private key; (2) generate the public key through the public key. The function $pair(m_1, m_2)$ is denoted by $(m_1, m_2)$ for short. The functions $AK\_Gen_1$ is denoted by $ak1$. And the function $BP_{Gen}$ is denoted by $bp$ that can be simplified, because the protocol using bilinear pairings satisfy BDDH assumption.

ProVerif shows that the compiled session with three participants is not a SGKE session. As a result, the compiled $(\alpha, \beta, \gamma)$-BBD protocol is not a SGKE protocol.

Then we revise the compiled $(\alpha, \beta, \gamma)$-BBD protocol as follow:

**Protocol Description of Revised $(\alpha, \beta, \gamma)$-BBD Protocol.** *Assume that $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic group of the same prime order $q$ with respective generators $g_1$ and $g_2$. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ such that $e(g_1, g_1) = g_2$. Assume that each participant has already generated a signature key $sk_i$ and the corresponding verification key $vk_i$. In addition, each participants has known the verification keys of others before executing the protocol, all the*

*indexes are taken modulo $n$, and $i \in [0, n-1]$. The protocol proceeds among all participants $\{p_i\}$ as follow:*
(1) *Each $p_i$ chooses a random number $x_i \in \mathbb{Z}_q^*$ and broadcasts it to all the others.*
(2) *Each $p_i$ chooses a random number $r_i \in \mathbb{Z}_q^*$, keeps $r_i$ secret, computes $Z_i = g_1^{r_i}$, and broadcasts $S_i = (Z_i, Sig_{sk_i}(Z_i, (p_0, x_0, p_1, x_1, p_2, x_2)))$*
(3) *Each $p_i$ verifies the messages $Verify(S_j, vk_j)$ that it has received. If all of them are valid, $p_i$ computes: $X_i = e((Z_{i-2}, Z_{i-1})^{\alpha r_i}) \cdot e((Z_{i-1}, Z_{i+1})^{\beta r_i}) \cdot e((Z_{i+1}, Z_{i+2})^{\gamma r_i}) = g_2^{\alpha r_{i-2} r_{i-1} r_i + \beta r_{i-1} r_i r_{i+1} + \gamma r_i r_{i+1} r_{i+2}}$, . Then broadcast $T_i = (X_i, Sig_{sk_i}(X_i, (p_0, x_0, p_1, x_1, p_2, x_2)))$.*
(4) *Each $p_i$ verifies the messages $Verify(T_j, vk_j)$ that it has received. If all of them are valid, $p_i$ computes: $k_i = g_2^{\sum_{i=0}^{n-1} r_i r_{i+1} r_{i+2}}$ Then $p_i$ erases $r_i$, let $k_i$ be the session key.*

Finally, we prove that the revised protocol is a SGKE protocol.

**Theorem 5.** *The revised $(\alpha, \beta, \gamma)$-BBD protocol is a SGKE protocol.*

We analyze the revised protocol by ProVerif. Based on Theorem 2 and Theorem 3, we only need to analyze a revised session with three participants as well. The specific program is shown in Fig 6 of Appendix.

ProVerif shows that the revised session with three participants is a SGKE session. Therefore, the revised $(\alpha, \beta, \gamma)$-BBD protocol is a SGKE protocol.

## 9. Conclusion

In this paper, we propose a fully automated approach to analyze group key exchange protocols that use bilinear pairings exchange and digital signatures without sacrificing the soundness of cryptography. When analyzing group key exchange protocols based on bilinear pairings and digital signatures with an unbounded number of sessions among arbitrary number of participants, researchers only need to provide the symbolic analysis of a single group key exchange session among three participants based on our work. Furthermore, the security is preserved when protocols are used to compose a more complex protocol by universal composition operation [10].

In addition, combined our approach with the work of [21], researchers can only design and prove the security of a single group key exchange session to resist outsider attack by the symbolic analysis, since it can be compiled to a secure protocol to resist malicious insider attack automatically.

Finally, there are two directions to extend our work:

(1) Researchers can analyze more group communication protocols that use bilinear pairings based on our work, such as ID-based group key exchange protocols, Joux Tripatite Diffie-Hellman protocol, etc.

(2) Note that we do not deal with key cycles. In other words, symbolic expressions are restricted to be acyclic. However, it may occur in some complex systems. Two possibility approaches to deal with key cycles are to strengthen the symbolic adversary or the symbolic security criterion.

*Acknowledgement*

This paper is supported by National Natural Science Foundation of China No.61003262, and National Natural Science Foundation of China No.60873237.

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[2] M. Abadi and B. Warinschi. Password-based encryption analyzed. In *the 32nd International Colloquium on Automata, Languages and Programming*, pages 664–676. LNCS 3580, Berlin: Springer-Verlag, 2005.

[3] P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness of formal encryption in the presence of key-cycles. In *the 10th European Symposium on Research in Computer Security*, pages 374–396. LNCS 3679, Berlin: Springer-Verlag, 2005.

[4] P. Adão, G. Bana, and A. Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *the 18th IEEE Computer Security Foundations Workshop*, pages 170–184. IEEE Computer Society, Pittsburgh, 2005.

[5] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *the 10th ACM Conference on Computer and Communications Security*, pages 220–230. Association for Computing Machinery, Washington, 2003.

[6] M. Backes and D. Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *the 21st IEEE Computer Security Foundations Symposium*, pages 255–269. IEEE Computer Society, Pittsburgh, 2008.

[7] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[8] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO 2008*, pages 108–125. LNCS 5157, Berlin: Springer-Verlag, 2008.

[9] E. Bresson, Y. Lakhnech, L. Mazaré, and B. Warinschi. A generalization of ddh with applications to protocol analysis and computational soundness. In *CRYPTO 2007*, pages 482–499. LNCS 4622, Berlin: Springer-Verlag, 2007.

[10] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *In 42nd Annual Syposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society, 2001.

[11] R. Canetti. Universally composable symbolic security analysis. *Journal of Cryptology*, 23(1):1–65, 2010.

[12] R. Canetti and S. Gajek. Universally composable symbolic analysis of diffie-hellman based key exchange. http://eprint.iacr.org/2010/303.pdf.

[13] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Cryptology-Eurocrypt 2001*, pages 453–474. LNCS 2045, Berlin: Springer-Verlag, 2001.

[14] R. Canetti and H. Krawczyk. Universally composable notions of key exchanged secure channels. In *Cryptology-Eurocrypt 2002*, pages 337–351. LNCS 2332, Berlin: Springer-Verlag, 2002.

[15] V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic system. *Journal of Automate Reasoning*, 46:225–259, 2011.

[16] D. Dolev and A. C. Yao. On the security of public key protocols. In *the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357. Nashville, TN, 1981.

[17] D. Galindo, F. D. Gracia, and P. van Rossum. Computational soundness of non-malleable commitments. In *the 4th Information Security Practice and Experience Conference*, pages 361–376. LNCS 4991, Berlin: Springer-Verlag, 2008.

[18] F. D. Gracia and P. van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Journal of Theoretical Computer Science*, 394:112–133, 2008.

[19] J. Herzog. A computational interpretation of dolevcyao adversaries. *Journal of Theoretical Computer Science*, 340:57–81, 2005.

[20] E. M. C. Jr., O. Grumberg, and D. A. Pe. *Model checking*, page 314. MIT Press, 1999.

[21] J. Katz and J. S. Shin. Modeling insider attacks on group key exchange protocols. In *the 12th ACM Conference on Computer and Communications Security*, pages 180–189. New York: ACM Press, 2005.

[22] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *CRYPTO 2003*, pages 110–125. LNCS 2729, Berlin: Springer-Verlag, 2003.

[23] S. Kremer and L. Mazaré. Computationally sound analysis of protocols using bilinear pairings. *Journal of Computer Security*, 18(6):999–1033, 2010.

[24] P. Laud and R. Corin. Sound computational interpretation of formal encryption with composed keys. In *the 6th International Conference on Information Security and Cryptology*, pages 55–66. LNCS 2971, Berlin: Springer-Verlag, 2004.

[25] D. Micciancio and B. Warinschi. Completeness theorems for the abadi-rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

[26] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *the 1st Theory of Cryptography Conference*, pages 133–151. LNCS 2951, Berlin: Springer-Verlag, 2004.

## Appendix: ProVerif Implementations

```
free c.
fun host/1.(* Participant identity *)
fun vk/1.(* Verication key of algebra A *)
fun pub/1.(* Public key of algebra A *)
fun ak1/2.(* Assistant key generate function of algebra A *)
fun bddh/3.(* Simplification based on BDDH assumption *)
data true/0.(* True evaluation *)
data zero/0. (* Round 1 *)
data one/0. (* Round 2 *)
data two/0. (* Round 3 *)
(* Property of bilinear pairing key *)
reduc bp(r2, (ak1(r0,(pub(r1),pub(r2))),ak1(r1,(pub(r2),pub(r0))))) = bddh(r2, pub(r0),pub(r1)).
equation bddh(r2, pub(r0),pub(r1)) = bddh(r0, pub(r1),pub(r2)).
equation bddh(r2, pub(r0),pub(r1)) = bddh(r1, pub(r2),pub(r0)).
fun sign/2. (* Signature function of algebra A *)
reduc checksign(vk(k), m, sign(k, m)) = true.(* Verify function of algebra A *)
let p0 = new r0; (* Process 0 *)
out(c,(host0, zero, r0)); (* Message 1 *)
in (c, recv0);
let (=host1, =zero, h1r) = recv0 in
in (c, recv1);
let (=host2, =zero, h2r) = recv1 in
new x0;
out(c, (host0, one, pub(x0), sign(sk0,(one,pub(x0),host0,r0,host1,h1r,host2,h2r))));(* Message 2 *)
in(c, recv2);
let (=host1, =one, d1, d1sig) = recv2 in
in(c, recv3);
let (=host2, =one, d2, d2sig) = recv3 in
if checksign(pk1,(one,d1,host0,r0,host1,h1r,host2,h2r), d1sig) = true then
if checksign(pk2,(one,d2,host0,r0,host1,h1r,host2,h2r), d2sig) = true then
out(c, (host0, two, ak1(x0,(d1,d2)), sign(sk0,(two,ak1(x0,(d1,d2)),host0,r0,host1,h1r,host2,h2r))));(* Mes-
sage 3 *)
in(c, recv4); in(c, recv5);
out(c,x0).(* Output internal state to the adversary *)
let p1 = new r1;(* Process 1 *)
out(c,(host1, zero, r1));(* Message 1 *)
in (c, recv0);
let (=host0, =zero, h0r) = recv0 in
in (c, recv1);
let (=host2, =zero, h2r) = recv1 in
new x1;
out(c, (host1, one, pub(x1), sign(sk1,(one,pub(x1),host0,h0r,host1,r1,host2,h2r))));(* Message 2 *)
in(c, recv2);
let (=host0, =one, d0, d0sig) = recv2 in
in(c, recv3);
let (=host2, =one, d2, d2sig) = recv3 in
if checksign(pk0,(one,d0,host0,h0r,host1,r1,host2,h2r), d0sig) = true then
if checksign(pk2,(one,d2,host0,h0r,host1,r1,host2,h2r), d2sig) = true then
out(c, (host1, two, ak1(x1,(d2,d0)), sign(sk1,(two,ak1(x1,(d2,d0)),host0,h0r,host1,r1,host2,h2r))));(* Mes-
sage 3 *)
in(c, recv4); in(c, recv5);
out (c, x1).(* Output internal state to the adversary *)
let p2 = new r2;(* Process 2 *)
out(c,(host2, zero, r2));(* Message 1 *)
in (c, recv0);
```

```
let (=host0, =zero, h0r) = recv0 in
in (c, recv1);
let (=host1, =zero, h1r) = recv1 in
new x2;
out(c, (host2, one, pub(x2), sign(sk2,(one,pub(x2),host0,h0r,host1,h1r,host2,r2))));(* Message 2 *)
in(c, recv2);
let (=host0, =one, d0, d0sig) = recv2 in
in(c, recv3);
let (=host1, =one, d1, d1sig) = recv3 in
if checksign(pk0,(one,d0,host0,h0r,host1,h1r,host2,r2), d0sig) = true then
if checksign(pk1,(one,d1,host0,h0r,host1,h1r,host2,r2), d1sig) = true then
out(c, (host2, two, ak1(x2,(d0,d1)), sign(sk2,(two,ak1(x2,(d0,d1)),host0,h0r,host1,h1r,host2,r2))));(* Mes-
sage 3 *)
in(c, recv4);
let (=host0, =two, e0, e0sig) = recv4 in
in(c, recv5);
let (=host1, =two, e1, e1sig) = recv5 in
if checksign(pk0,(two,e0,host0,h0r,host1,h1r,host2,r2), e0sig) = true then
if checksign(pk1,(two,e1,host0,h0r,host1,h1r,host2,r2), e1sig) = true then
out(c, x2);(* Output internal state to the adversary *)
out(c, choice[bp(x2,(e0,e1)), bp(rc,(ak1(ra,(pub(rb),pub(rc))),ak1(rb,(pub(rc),pub(ra)))))]).(* output a
real or random key *)
process
new sk0; let pk0 = vk(sk0) in
new sk1; let pk1 = vk(sk1) in
new sk2; let pk2 = vk(sk2) in
let host0 = host(sk0) in
let host1 = host(sk1) in
let host2 = host(sk2) in
new ra; new rb; new rc;
(p0 | p1 | p2)
```

Fig 5. The ProVerif Specification of Compiled $(\alpha, \beta, \gamma)$-BBD Protocol with respect to Forward Secrecy.

```
free c.
fun host/1.(* Participant identity *)
fun vk/1.(* Verication key of algebra A *)
fun pub/1.(* Public key of algebra A *)
fun ak1/2.(* Assistant key generate function of algebra A *)
fun bddh/3.(* Simplification based on BDDH assumption *)
fun erase/1.(* Erase the private key *)
data true/0.(* True evaluation *)
data zero/0. (* Round 1 *)
data one/0. (* Round 2 *)
data two/0. (* Round 3 *)
(* Property of bilinear pairing key *)
reduc bp(r2, (ak1(r0,(pub(r1),pub(r2))),ak1(r1,(pub(r2),pub(r0))))) = bddh(r2, pub(r0),pub(r1)).
equation bddh(r2, pub(r0),pub(r1)) = bddh(r0, pub(r1),pub(r2)).
equation bddh(r2, pub(r0),pub(r1)) = bddh(r1, pub(r2),pub(r0)).
fun sign/2. (* Signature function of algebra A *)
reduc checksign(vk(k), m, sign(k,m)) = true.(* Verify function of algebra A *)
let p0 = new r0; (* Process 0 *)
out(c,(host0, zero, r0)); (* Message 1 *)
in (c, recv0);
let (=host1, =zero, h1r) = recv0 in
in (c, recv1);
let (=host2, =zero, h2r) = recv1 in
new x0;
out(c, (host0, one, pub(x0), sign(sk0,(one,pub(x0),host0,r0,host1,h1r,host2,h2r))));(* Message 2 *)
in(c, recv2);
let (=host1, =one, d1, d1sig) = recv2 in
in(c, recv3);
let (=host2, =one, d2, d2sig) = recv3 in
if checksign(pk1,(one,d1,host0,r0,host1,h1r,host2,h2r), d1sig) = true then
if checksign(pk2,(one,d2,host0,r0,host1,h1r,host2,h2r), d2sig) = true then
out(c, (host0, two, ak1(x0,(d1,d2)), sign(sk0,(two,ak1(x0,(d1,d2)),host0,r0,host1,h1r,host2,h2r))));(* Mes-
sage 3 *)
in(c, recv4); in(c, recv5);
out(c,erase(x0)).(* Output internal state to the adversary *)
let p1 = new r1;(* Process 1 *)
out(c,(host1, zero, r1));(* Message 1 *)
in (c, recv0);
let (=host0, =zero, h0r) = recv0 in
in (c, recv1);
let (=host2, =zero, h2r) = recv1 in
new x1;
out(c, (host1, one, pub(x1), sign(sk1,(one,pub(x1),host0,h0r,host1,r1,host2,h2r))));(* Message 2 *)
in(c, recv2);
let (=host0, =one, d0, d0sig) = recv2 in
in(c, recv3);
let (=host2, =one, d2, d2sig) = recv3 in
if checksign(pk0,(one,d0,host0,h0r,host1,r1,host2,h2r), d0sig) = true then
if checksign(pk2,(one,d2,host0,h0r,host1,r1,host2,h2r), d2sig) = true then
out(c, (host1, two, ak1(x1,(d2,d0)), sign(sk1,(two,ak1(x1,(d2,d0)),host0,h0r,host1,r1,host2,h2r))));(* Mes-
sage 3 *)
in(c, recv4); in(c, recv5);
out (c, erase(x1)).(* Output internal state to the adversary *)
let p2 = new r2;(* Process 2 *)
out(c,(host2, zero, r2));(* Message 1 *)
in (c, recv0);
let (=host0, =zero, h0r) = recv0 in
```

```
in (c, recv1);
let (=host1, =zero, h1r) = recv1 in
new x2;
out(c, (host2, one, pub(x2), sign(sk2,(one,pub(x2),host0,h0r,host1,h1r,host2,r2))));(* Message 2 *)
in(c, recv2);
let (=host0, =one, d0, d0sig) = recv2 in
in(c, recv3);
let (=host1, =one, d1, d1sig) = recv3 in
if checksign(pk0,(one,d0,host0,h0r,host1,h1r,host2,r2), d0sig) = true then
if checksign(pk1,(one,d1,host0,h0r,host1,h1r,host2,r2), d1sig) = true then
out(c, (host2, two, ak1(x2,(d0,d1)), sign(sk2,(two,ak1(x2,(d0,d1)),host0,h0r,host1,h1r,host2,r2))));(* Mes-
sage 3 *)
in(c, recv4);
let (=host0, =two, e0, e0sig) = recv4 in
in(c, recv5);
let (=host1, =two, e1, e1sig) = recv5 in
if checksign(pk0,(two,e0,host0,h0r,host1,h1r,host2,r2), e0sig) = true then
if checksign(pk1,(two,e1,host0,h0r,host1,h1r,host2,r2), e1sig) = true then
out(c, erase(x2));(* Output internal state to the adversary *)
out(c, choice[bp(x2,(e0,e1)), bp(rc,(ak1(ra,(pub(rb),pub(rc))),ak1(rb,(pub(rc),pub(ra)))))]).(* output a
real or random key *)
process
new sk0; let pk0 = vk(sk0) in
new sk1; let pk1 = vk(sk1) in
new sk2; let pk2 = vk(sk2) in
let host0 = host(sk0) in
let host1 = host(sk1) in
let host2 = host(sk2) in
new ra; new rb; new rc;
(p0 | p1 | p2)
```

Fig 6. The ProVerif Specification of Revised $(\alpha, \beta, \gamma)$-BBD Protocol with respect to Forward Secrecy.