A Reflection on the Security of Two-Party Key Establishment Protocols

Qiang Tang
DIES, Faculty of EEMCS
University of Twente, the Netherlands
q.tang@utwente.nl

Abstract. Two-party key establishment has been a very fruitful research area in cryptography, with many security models and numerous protocols proposed. In this paper, we take another look at the YAK protocol and the HMQV protocols and present some extended analysis. Motivated by our analysis, we reflect on the security properties that are desired by two-party key establishment protocols, and their formalizations. In particular, we take into account the interface between a key establishment protocol and the applications which may invoke it, and emphasize the concept of session and the usage of session identifier. Moreover, we show how to design a two-party key establishment protocol to achieve both key authentication and entity authentication properties in our security model.

Keywords: key establishment, key authentication, entity authentication

1 Introduction

The history of two-party key establishment goes back a long way, although the modern study of key establishment protocols can be traced back to the seminal work of Needham and Schroeder [24]. Particularly, since the seminal work of Diffie and Hellman [11], key establishment has been a very fruitful area in cryptography. So far, numerous protocols have been proposed, as surveyed in Chapter 3 of Tang’s PhD thesis [27] and the book by Boyd and Mathuria [5]. The standardization bodies, such as ISO and IEEE, have also created a number of key establishment standards [15,16,17,18,19]. Among these protocols, a large proportion are based on the concept of Diffie-Hellman key exchange [11].

Before the pioneering work by Bellare and Rogaway [3], who first analysed key establishment protocols using complexity-theoretic methods, the security of key establishment protocols was typically evaluated using only heuristic techniques. As a result, many protocols have been proposed which contain subtle vulnerabilities, only discovered after the schemes were published. The Bellare-Rogaway model [3] was designed to enable the analysis of entity authentication and two-party key establishment in the shared secret key setting. Subsequently, a number of variants of this model have been proposed. Blake-Wilson, Johnson, and Menezes [4] extended the model to the public key setting. Bellare, Canetti, and Krawczyk [2] provided a modular approach
for the construction of authenticated key establishment protocols. They also proposed the first simulatability-based security model for key establishment. Shoup [25] refined the simulatability-based security model and proposed a model which works under three different corruption assumptions. Later, Bellare, Canetti, and Krawczyk [26] further extended the concept, and proposed another security model for two-party key establishment protocols in the universally composable security framework. Hitchcock, Boyd, and Nieto [14] optimised the Bellare-Canetti-Krawczyk model. A number of papers have been devoted to discussing the validation of, and relationships between, these various security models, e.g. [789].

1.1 Motivation and Contribution

In the literature, there have been a lot attacks on two-party key establishment protocols. Some of them are feasible because the target protocols lack rigorous security analysis, while others are feasible because they fall beyond the security model or are carried out in some unperceived execution scenarios. Motivated by some recent attacks against the provably secure HMQV protocols [13], we consider it to be an interesting task to reflect on the security of two-party key establishment protocols.

In this paper, we revisit two-party key establishment protocols, which exploit asymmetric key cryptography for security guarantees. This category of two-party protocols is of interest to us, because it provides better scalability than the category of protocols using symmetric key cryptography for security guarantees while provides stronger security than the category of protocols using passwords for security guarantees. Specifically, our contribution can be summarized as follows.

1. First, we take another look at the YAK protocol proposed by Feng [13] and HMQV protocols [20,21] and present some extended analysis. In particularly, we show that, contradicting Feng’s claim, the YAK protocol does not achieve forward secrecy. The YAK protocol and the HMQV protocols do not achieve unknown key share resilience in some execution scenarios. In addition, we show that the 3-pass HMQV protocol is vulnerable to a key compromise impersonation attack, when the adversary has access to a user’s long-term and short-term private keys.

2. Second, we re-examine two-party key establishment protocols from a number of aspects: the workflow, the involved secrets, the interface with invoking applications. In particular, we emphasize that the invoking application should provide application identifier to the key establishment protocol, which should return a (session identifier, session key) pair. To our knowledge, such extended examination has not been done in the literature of two-party key establishment protocols.

3. Third, we categorize the security properties into two categories, namely key authentication and entity authentication, and formalize them accordingly. The key authentication property guarantees that only the intended user is capable of computing a session key, while the entity authentication property guarantees that the intended user has actually involved in a session. Both properties are crucial for enabling a security service by running a key establishment protocol. Most existing attack scenarios have been covered in our security model.
4. Last, we present a two-party key establishment protocol which achieves both key authentication and entity authentication properties in our security model.

Arguably, by requiring the underlying applications to adopt some extra security mechanisms, a protocol proven secure in any existing security model could possibly achieve the same level of security in practice as a protocol proven secure in our security model. However, we believe it is important to use a comprehensive security model without relying on the underlying applications.

1.2 Organization

The rest of the paper is organised as follows. In Section 2, we analyse the YAK protocol and the HMQV protocols. In Section 3, we look at different aspects of two-party key establishment protocols. In Section 4, we categorize the security properties for two-party key establishment protocols and formalize them. In Section 5, we present a two-party key establishment protocol which is secure in our security model. In Section 6, we conclude the paper.

2 Review of YAK and HMQV

Throughout the paper, we use $x \in_R X$ to denote that $x$ is chosen from the set $X$ uniformly at random, and use $x||y$ to denote the concatenation of $x$ and $y$.

2.1 Review of the YAK Protocol

The YAK protocol, proposed by Feng [13], is a two-party key establishment protocol. Note that this protocol has not been rigorously analysed in a security model. Let Alice and Bob be two users who trust TTP in common. The algorithms are defined as follows.

1. System Setup: Alice and Bob agree on a group $G$ of prime order $q$, a generator $g$ of $G$, and a hash function $H$. Alice selects $a \in_R \mathbb{Z}_q$ as her private key. Alice sends $g^a$ to the TTP with a proof about her knowledge on $a$, and receives a certificate on her public key $g^a$ from the TTP if the proof is valid. The proof of knowledge is a non-interactive protocol described in [23], defined as follows.
   (a) Alice sends $X = g^a$ and $\{\text{Alice, OtherInfo, } V = g^v, r = v - a \cdot h\}$ to the TTP, where $v \in_R \mathbb{Z}_q$, $h = H(g, V, X, \text{“Alice”, OtherInfo})$, and OtherInfo includes some auxiliary information.
   (b) The TTP checks $X$ has the prime order $q$ and $V = g^r X^h$.

In a similar way, Bob generates his private key $b \in_R \mathbb{Z}_q$ and obtains a certificate on his public key $g^b$ from the TTP.
2. Key establishment: The key establishment protocol is as follows.

(a) Alice selects $x \in R \mathbb{Z}_q$, and sends $g^x$ and $\text{prf}_1$ to Bob. The $\text{prf}_1$ is a proof of knowledge of $x$, where

$$\text{prf}_1 = \{ \text{“Alice”}, \text{OtherInfo}_1, V_1 = g^{v_1}, r_1 = v_1 - x \cdot h_1 \},$$

$v_1 \in R \mathbb{Z}_q$, $h_1 = H(g, V_1, g^x, \text{“Alice”}, \text{OtherInfo}_1)$.

In the meantime, Bob selects $y \in R \mathbb{Z}_q$, and sends $g^y$ and $\text{prf}_2$ to Alice. The $\text{prf}_2$ is a proof of knowledge of $y$, where

$$\text{prf}_2 = \{ \text{“Bob”}, \text{OtherInfo}_2, V_2 = g^{v_2}, r_2 = v_2 - y \cdot h_2 \},$$

$v_2 \in R \mathbb{Z}_q$, $h_2 = H(g, V_2, g^y, \text{“Bob”}, \text{OtherInfo}_2)$.

$\text{OtherInfo}_1$ and $\text{OtherInfo}_2$ include some auxiliary information.

(b) If Alice accepts Bob’s proof, she computes the session key as $H((g^y \cdot g^b)^{a+x})$. Similarly, if Bob accepts Alice’s proof, he computes the session key as $H((g^x \cdot g^a)^{b+y})$.

**Vulnerability against forward secrecy.** From the description, we note that the key establishment protocol is one round and the proof of knowledge of ephemeral private key is non-interactive. In addition, Bob is not required by Alice to prove his knowledge of the long-term private key $b$. This results in the fact that, in a session, an adversary can select $r \in R \mathbb{Z}_q$, and send $g^r$ and $\text{prf}_2$ to Alice. The $\text{prf}_2$ is a proof of knowledge of $r$, where

$$\text{prf}_2 = \{ \text{“Bob”}, \text{OtherInfo}_2, V_2 = g^{v_2}, r_2 = v_2 - r \cdot h_2 \},$$

$v_2 \in R \mathbb{Z}_q$, $h_2 = H(g, V_2, g^r, \text{“Bob”}, \text{OtherInfo}_2)$.

Clearly, Alice will accept the adversary’s message and generate a session key $K = H((g^r \cdot g^b)^{a+x})$. Consequently, we have the following attack scenario.

*Suppose Alice has initiated the key agreement process to generate a session key to send Bob some data $M$. Then, after the key establishment process successfully finishes, Alice may just send $\text{Encrypt}(M, K)$ to Bob and end the session afterwards, where $\text{Encrypt}$ is a symmetric key encryption algorithm. Later on, at any time, if the adversary can corrupt Bob’s long-term private key, it can generate the same session key $K$ and recover $M$.**

In the above attack scenario, Alice has in fact established a session key with the adversary, instead of with Bob who has actually not been involved at all in the key establishment process. Moreover, Bob is not able to compute $K$ since he has no knowledge about the ephemeral secret $r$. This attack violates the forward secrecy property, as defined in Section 4. In fact, this attack coincides with Krawczyk’s statement that 2-pass key establishment protocol authenticated via public keys and with no secure shared state previously established between the parties will be vulnerable to the attacks against forward secrecy.
**Vulnerability against unknown key share resilience.** Suppose Alice has two devices, say Laptop and Workstation, and two applications which result in communications between these two devices. Consider a scenario where the applications run the YAK protocol to establish session keys to protect the communications simultaneously. In the absence of attacks, the protocol executions will perform as in Fig. 1.

![Fig. 1. Concurrent Sessions of YAK](image)

In more details, the four knowledge proofs, namely $prf_1^i$ (1 ≤ $i$ ≤ 2) and $prf_2^i$ (1 ≤ $i$ ≤ 2) are defined as follows.

- $prf_1^i = \{\text{Laptop, OtherInfo}_1^i, V_1' = g^{v_1'}, r_1' = v_1' - x_1' \cdot h_1'\}$, where $v_1' \in R Z_q$, $h_1' = H(g, V_1', g^{x_1'}, \text{Laptop, OtherInfo}_1^i)$.
- $prf_2^i = \{\text{Workstation}, OtherInfo}_2^i, V_2' = g^{v_2'}, r_2' = v_2' - y_2' \cdot h_2'\}$, where $v_2' \in R Z_q$, $h_2' = H(g, V_2', g^{y_2'}, \text{Workstation, OtherInfo}_2^i)$.
- $prf_1'' = \{\text{Laptop, OtherInfo}_1^'', V_1'' = g^{v_1''}, r_1'' = v_1'' - x_1'' \cdot h_1''\}$, where $v_1'' \in R Z_q$, $h_1'' = H(g, V_1'', g^{x_1''}, \text{Laptop, OtherInfo}_1^{'})$.
- $prf_2'' = \{\text{Workstation, OtherInfo}_2^'', V_2'' = g^{v_2''}, r_2'' = v_2'' - y_2'' \cdot h_2''\}$, where $v_2'' \in R Z_q$, $h_2'' = H(g, V_2'', g^{y_2''}, \text{Workstation, OtherInfo}_2^{'})$.

With respect to the above descriptions, we note that Laptop’s proofs, namely $prf_1^i$ and $prf_2^i$, are generated based on the same identity Laptop. Similarly, we note that Workstation’s proofs, namely $prf_2^i$ and $prf_2''$, are generated based on the same identity Workstation. Therefore, an adversary can swap messages in the two concurrent sessions and mount an unknown key share attack. The attack is shown in Fig. 2.

It is straightforward to verify that Laptop’s session 1 and Workstation’s session 2 will compute the same session key, namely $H((g^{x_1'} \cdot g^{x_1''})^{y_2'' + y_2'})$. While, Laptop’s session 2 and Workstation’s session 1 will compute the same session key, namely $H((g^{y_2'} \cdot g^{y_2''})^{x_1'' + x_1'})$.

The consequence of the attack is that the adversary can swap messages in the two applications without being noticed. For example, if in application I, Laptop sends $\text{Encrypt}(M, K)$ to Workstation where $\text{Encrypt}$ is a symmetric key encryption algorithm, then the adversary can forward it to Workstation in application II which will successfully decrypt the ciphertext. As a result, messages have been delivered to the unintended recipient applications, and this may cause subtle attacks depending on the applications.
2.2 Review of the HMQV Protocols

The HMQV family consists of a number of protocols, and we describe the 2-pass and 3-pass protocols below. Let Alice and Bob be two users who trust TTP in common. Alice and Bob agree on a group $G$ of prime order $q$, a generator $g$ of $G$, and a hash function $H$. Alice selects her long-term private key $a \in \mathbb{Z}_q$ and lets the TTP certify the public key $g^a$, and Bob selects his long-term private key $b \in \mathbb{Z}_q$ and lets the TTP certify the public key $g^b$. The protocols are shown in Fig. 3.

**Fig. 3.** The HMQV Protocols

The value $\sigma_a$ and $\sigma_b$ are defined as follows.

$$d = \overline{H}(X||"Bob"), e = \overline{H}(Y||"Alice"), \sigma_a = (Yg^be)x+da, \sigma_b = (Xg^{ad})y+eb,$$

where $\overline{H}$ outputs the first $\ell$ bit of the input of the hash function $H$ given that the security parameter is $\ell$. It is straightforward to verify that

$$\sigma_a = \sigma_b = g^{(x+ad)(y+be)}.$$

Menezes [22] showed that both protocols might be vulnerable to so called “small subgroup attacks” and “lattice attacks”, through which Alice, if being malicious,
could obtain Bob’s long-term private key \( b \). These attacks have exploited the fact that, in the protocol specification, it is not mandatory for Alice and Bob to verify that both long-term and short-term keys are generated from the right subgroup. Feng [13] showed that the 2-pass protocol is vulnerable to unknown share attacks if the same user executes the protocol between different devices. Furthermore, we have the following observations.

1. We show a key compromise impersonation attack in which, if the adversary knows Alice’s long-term and ephemeral private keys \( a \) and \( x \), it can impersonate Bob to Alice. We skip the details since it is straightforward to verify that the attack works.

\[
\begin{align*}
\text{Alice} & (a, g^a) \\
\text{Adversary} & (a', g^{a'}, b, x) \\
\end{align*}
\]

\[
\begin{align*}
x \in \mathbb{Z}_q \\
X = g^x \\
y \in \mathbb{Z}_q, Y = g^y, \sigma_b = g^{(x+ad)y} \cdot (g^{b})^{(x+ad)e} \\
\sigma_a = H(\sigma_b || 0) \\
k_m = H(\sigma_a || 0) \\
Z = \text{MAC}(\sigma_a || 1, k_m) \\
Y, Z \\
W = \text{MAC}(\sigma_a || 0, k_m) \\
W \\
sk = H(\sigma_a || 1) \\
sk = H(\sigma_b || 1)
\end{align*}
\]

Fig. 4. Attack against the 3-pass HMQV Protocol

2. Both protocols are vulnerable to the type of unknown key share attacks shown in the case of the YAK protocol. Nonetheless, we note that this kind of attack is beyond the security model used in [20,21].

3. Feng [13] presented a so called ”invalid public key attack”. To mount the attack, Bob lets the TTP certify a long-term public key without generating any private key. Due to the deliberate selection of his public key, Bob can still successfully establish a session key with Alice. Since the public key is weak, any entity can impersonate Bob to Alice so that it causes authentication and repudiation “problems”. However, this kinds of problems apply to all protocols, if a user does not follow the protocol specification and chooses weak keys on purpose.

4. An execution of the 3-pass protocol requires 3 sequential message exchanges: Bob can only send his first message after he receives Alice’s first message, Alice can only send her second message after she receives Bob’s first message, and Bob can only end the execution after he receives Alice’s second message. The process can be improved by turning it into a 4-pass protocol which requires only 2 sequential message exchanges. To achieve this, Alice and Bob first simultaneously send \( X \) and \( Y \) to each other, then simultaneously send \( Z \) and \( W \) to each other.
3 Overview of Two-Party Key Establishment

3.1 The workflow

A two-party key establishment protocol generally involves two types of entities.

- One type is users, any two of which may run the key establishment protocol to establish a session key with each other. Throughout the paper, we assume that the users that may run a key establishment protocol are denoted as \( U_i \) (\( 1 \leq i \leq N \)), where \( N \) is the maximum of user population. Furthermore, we assume that \( U_i \) possesses the identity \( \text{ID}_i \). If \( i \neq j \), \( U_i \) and \( U_j \) are different users.
- The other type of entity is a Trusted Third Party (TTP) which help users validate their long-term public keys, usually of some public-key encryption or digital signature schemes.

Generally, we consider the following execution scenarios for a two-party key establishment protocol.

1. Two users, say \( U_i \) and \( U_j \) (\( i \neq j \)), may use the same protocol and the same long-term key to negotiate session keys for many applications. The applications may reside in different devices of \( U_i \) and \( U_j \). For example, they may run the protocol to between their laptops to protect their chatting contents and between their work PCs to protect their work documents.
2. The same user, say \( U_i \), may have multiple devices, any two of which use the same protocol and the same long-term key to negotiate session keys for for many applications. For example, Alice may have separate applications to remotely transfer medical files and sensitive work documents from her laptop to her PC in her company.
3. For each application, the protocol may be executed multiple times to establish cryptographic keys. We denote each execution of the protocol to be a session. We further assume that multiple sessions can be initiated simultaneously.

We use \( \mathcal{S} \) to denote the application identifier set which contains the application identifiers of all users.

In our setting, a session denotes an execution of the key establishment protocol, invoked by an application of two users. Informally, a session is associated with a set \( \{\text{ID}_i, \text{ID}_j, \text{appid}\} \) and some execution-specific information (such as the message exchange transcript) which distinguish different protocol executions invoked by the same application of the two users. Formally, we use a session identifier to identify a session, and formally define its property under Definition \( \Box \) in Section \( \Box \). This concept is important not only for theoretically formalizing the security of a key establishment protocol, but also for applications to appropriately use the session keys.
**Remark 1.** In fact, the concept of *session* has been used in the existing models, e.g. [3,6]. However, it has not been advocated that a session identifier should be outputted to the invoking application, together with the session key, e.g. in [3]. In [6], it assumed that a session identifier should be supplied by the invoking application as an input, however, this will require the two participants of an application to negotiate an identifier in advance securely. Arguably, this is an undesirable assumption.

![Diagram](image-url)

**Fig. 5.** Workflow of a Two-party Key Establishment Protocol

The workflow of a two-party key establishment protocol follows the style shown in Fig. 5. The inclusion of identifiers $\text{ID}_i$ and $\text{ID}_j$ in the input to the key establishment protocol states that the users know their intended partners, and this is always implicitly assumed in other works. Here, we have two new features.

1. One is that the key establishment protocol takes an application identifier $\text{appid}$ as input. This reflects our consideration that the protocol can be invoked by multiple applications from the same user. We require that, for a user, its different applications hold different identifiers. Furthermore, if an application is between $U_i$ and $U_j$ (say, transferring files between these users), then these two users hold the same identifier for the application.

2. The key establishment protocol returns a (session identifier, session key) pair, instead of a session key only as in most existing protocols.

These features are used to distinguish the key establishment sessions invoked by different applications, and subsequently they enable us to avoid the type of unknown key share attacks shown in Section 2.

**Remark 2.** About the protocol execution, we also make the following assumptions. Both $U_i$ and $U_j$ know how to validate the public keys of each other. At the beginning of a session, it may be required that $U_i$ sends $m_1$ to $U_j$ first before $U_j$ sends $m_2$ back, namely there is an order of message exchange. In such case, we say $U_i$ is the *initiator* while $U_i$ is the *responder*. If there is no such order requirement, we simply say both users are *initiator*.
3.2 The long-term and ephemeral private keys

In a session of a two-party key establishment protocol, each participant makes use of two types of secrets.

1. One is its long-term private key, which guarantees the integrity and/or confidentiality of messages sent to the other participant. In some protocols, the long-term private key is also used to derive the session key. Without using a long-term private key, a two-party key establishment protocol will suffer from a man-in-the-middle attack, for example in the case of Diffie-Hellman.

2. The other is the ephemeral private key, which is the randomness locally generated by a random number generator or a pseudo-random number generator. The ephemeral private key is mainly used to derive a fresh session key, and it makes the session keys unique.

In practice, both types of private keys could be compromised. For the long-term private key, the user must store it somewhere and may also make some backups. Therefore, an adversary may obtain the long-term private key if it gains access to any one of the storage devices. With respect to the ephemeral private key, since it is generated locally by a computing device (normally a computer), an adversary may gain access to it by a number of means. For example, the adversary could install some trojan horse in the participant’s computing device to steal the secrets, or it could make use of the imperfectness of the pseudo-random number generator in use.

In practice, a user may attempt to use its long-term private key for other purposes. For example, a user may use the signature key pair (assuming such a key pair is used in the key establishment protocol) to authenticate itself to a website. This kind of extended usage of the same private key is extremely risky. To demonstrate our point, we present a toy example of a key establishment protocol in Fig. 6 and a toy authentication protocol in Fig. 7, where $r$ is a random string, $(pk_2, sk_2)$ is a sign/verify key pair of $U_2$.

![Fig. 6. Key Establishment Protocol](image)

![Fig. 7. Authentication Protocol](image)

Since $U_1$ can make $U_2$ sign a nonce in the key establishment protocol, so that $U_1$ can impersonate $U_2$ in the authentication protocol trivially. We emphasize that

---

1 The same long-term private key is used in all executions of the protocol.
extended usage of the same private key should be avoided in practice. To formulate
the security properties, we assume the user’s long-term private key is only used in
executing a two-party key establishment protocol.

4 Formalization of Security Properties

In this section, we formalize the security properties desired by a two-party key estab-
lishment protocol. Further remarks are provided in the Appendix A.

4.1 The threat model

With respect to the relevant trust relationships in a two-party key establishment
protocol, we make the following assumptions.

1. If a TTP is required, it is semi-trusted in the following sense. The TTP will
honestly follow the protocol specification. For example, it will honestly certify
users’ public keys, but will not generate a forged certificate for either an existing
identity or a new identity (which is different from $\text{id}_i$ ($1 \leq i \leq N$)).

2. Any user, say $U_i$, is semi-trusted by any other user $U_j$ ($j \neq i$) in the following
sense.

   (a) Suppose $U_j$ is engaged in a session with $U_i$, it may be possible that $U_i$ will
deviate from the protocol specification in order to obtain more information
about $U_j$’s long-term and short-term private keys. For example, $U_i$ may try
to impersonate $U_j$ to another user in another session, by taking advantage of
the harvested information.

   (b) Suppose $U_j$ is engaged in a session with $U_i$, besides the possible malicious
activities described in (a), $U_i$ will do nothing else malicious. For example, $U_i$
will not disclose the shared session key with $U_j$ to any other entity.

   (c) Suppose $U_j$ is engaged in a session with $U_k$, $U_i$ is treated as a malicious
adversary.

Since a key establishment protocol is normally used in a distributed environment,
namely the two participants communicate through an open network. Therefore, it is
essential to assume the adversary to be an active one. In other words, the adversary
can not only eavesdrop on the communication network but also manipulate (delete,
insert, replace, delay, etc) the messages at its will.

As a basic requirement, a two-party key establishment protocol should always be
sound. Formally the soundness property is defined as follows.

\textbf{Definition 1.} A two-party key agreement protocol is sound if the following require-
ments are satisfied.
1. In the absence of an adversary, the two users obtain the same (session identifier, session key) pair at end of a session.
2. For any two sessions of $U_i$, carried out with $U_j$ for an application $app_i d$, the probability that the session identifiers are identical is negligible, regardless of the coin flips of $U_j$.
3. Suppose that $U_i$ successfully ended a session with $U_i$ for the application $app_i d$ and $U_k$ successfully ended a session with $U_i$ for the application $app'_i d$. Given that the sets $\{ID_i, ID_j, app_i d\}$ and $\{ID_k, ID_i, app'_i d\}$ are not identical, the probability that the session identifier of $ID_i$ and the session identifier of $ID_k$ are identical is negligible, regardless of the coin flips of all users.

In the literature, many properties have been identified for key agreement protocols, and a summary of them can be found in the work by Menezes, Oorschot, and Vanstone [23], Boyd and Mathuria [5], and Tang [27]. In this paper, we categorize the desirable properties for a key establishment protocol into two categories, namely key authentication and entity authentication.

### 4.2 The Key Authentication Property

The Key authentication property[2] states that a user $U_i$ has completed a successful protocol execution in one session (let it be denoted with session identifier $sid^*$) with $U_j$, it can be assured that only $U_j$ is capable of compute the same session key. It is worth noting that this property does not guarantee that $U_j$ actually possess the same session key. In fact, it does not even guarantee that $U_j$ actually knows there is such a key establishment session.

When analysing the key authentication property, in the above threat model, we need to assume an active adversary which can access to the long-term private keys of all users except $U_i$ and $U_j$. In fact, we often need to assume a more powerful adversary in the following attack scenarios.

1. **known session key attack**: In this case, the adversary is assumed to be able to compromise the session keys in all sessions except that identified by $sid^*$. The concept was first mentioned by Denning and Sacco [10] in a limited way, in which the adversary can compromise all past session keys.
2. **forward secrecy attack**: In this case, the adversary is assumed to be able to compromise the long-term private keys of $U_i$ or $U_j$ after the session with identifier $sid^*$ has ended. If both private keys are compromised by the adversary, it is said to be a perfect forward secrecy attack.
3. **known master key attack**: In this case, the adversary is assumed to be able to compromise TTP and obtain its long-term private keys. In addition, if the TTP stores also other secret information then the adversary can obtain it as well.
4. **unknown key-share attack**: The notion of unknown key-share attacks were first discussed by Diffie, van Oorschot and Wiener [12]. It is defined to be the property that an honest user $U_i$ never ends up believing it shares a key with user $U_j$.

---

[2] It is also referred to as the key secrecy property in the literature.
although it actually shares the session key with another user \( U_u \) \((u \neq j)\). We further require that one application should not share the same the session key with another application when the protocol is run by the two applications of the same pair of users.

5. **key-compromise impersonation attack**: This attack makes sense only when \( i \neq j \), namely the two users in the session are different. In this case, the adversary is assumed to be able to compromise the long-term private key of \( U_i \). Note that this attack scenario is also considered in evaluating the entity authentication property.

Formally, the key authentication property is evaluated by the attack game between a challenger and an adversary, as shown in Fig. 8, where the adversary’s advantage is defined to be \( |\Pr[b = b' | \cdot \frac{1}{2}]\). For the simplicity of description, we use the notations \( U_i \) for some \( 1 \leq i \leq N \) in a few places, although the challenger will actually simulate all these activities.

**Definition 2.** A two-party key establishment protocol achieves the key authentication property, if any polynomial-time adversary has only negligible advantage in the key authentication game.

### 4.3 The Entity Authentication Property

The entity authentication property states that, if from the view of a user \( U_j \), it has successfully completed a key establishment session (identified by the session identifier \( sid^* \)) with \( U_j \), then \( U_j \) should have actually engaged in a key establishment session (identified by the session identifier \( sid^* \)) with \( U_i \). Typically, we need to assume an active adversary which can access to the long-term private keys of all users except \( U_i \) and \( U_j \). Similar to the situation in analysing the key authentication property, we often also consider some extended attack scenarios.

1. **key-compromise impersonation attack**: This attack makes sense only when \( i \neq j \), namely the two users in the session are different. In this case, the adversary is assumed to be able to compromise the long-term private key of \( U_i \). In addition, we also allow the adversary to compromise all the ephemeral secrets of \( U_i \).

2. **key confirmation**: In this case, if, from the view of a user \( U_i \), it has successfully completed a key establishment session (identified by the session identifier \( sid^* \)) with \( U_j \), then \( U_j \) should have engaged in a key establishment session (identified by the session identifier \( sid^* \)) with \( U_i \) and has computed the same session key as that of \( U_i \).

Formally, the entity authentication property is evaluated by the attack game between a challenger and an adversary, as shown in Fig. 9, where the advantage is defined to be the probability that the user \( U_i \) does not have a session with the identifier \( sid^* \) and session key \( sk^* \). For the simplicity of description, we use the notations \( U_i \) for some \( 1 \leq i \leq N \) in a few places, although the challenger will actually simulate all these activities.
1. Setup: the challenger generates the parameters for the TTP and publishes the public parameter.

2. Phase 1: Besides delivering messages for all sessions, the adversary is allowed to issue the following types of queries for any $1 \leq i, j \leq N$.
   (a) $\text{Register}_u(U_i)$: The user $U_i$ selects its identity $\text{ID}_i$, generates a key pair $(PK_i, SK_i)$, and let the challenger certify $PK_i$.
   (b) $\text{Register}_u(U_j)$: The adversary selects an identity $\text{ID}_j$, generates a key pair $(PK_i, SK_i)$, and let the challenger certify $PK_i$.
   (c) $\text{Invoke}(U_i, U_j, app_id, \text{role})$: The user $U_i$ initiates a new session with $U_j$ for an application identified by $app_id \in S$. We require that both users have registered their public keys. The session is locally indexed by $(L_i, \text{ID}_i, \text{ID}_j, app_id)$, where the value $L_i$ means that it is $U_i$’s $L_i$-th session. In addition, $U_i$ does the following:
   - If role = initiator, $U_i$ starts sending a message to $U_j$ by following the protocol specification.
   - If role = responder, $U_i$ waits for $U_j$’s message by following the protocol specification.

   (d) $\text{Corrupt}_u(L_i, \text{ID}_i)$: The user $U_i$ sends the ephemeral secrets of its $L_i$-th session to the adversary.

   (e) $\text{Corrupt}_u(\text{ID}_i)$: The user $U_i$ sends its long-term private key $SK_i$ to the adversary.

   (f) $\text{Corrupt}_u(L_i, \text{ID}_i)$: If the $U_i$’s $L_i$-th session has successfully ended, which means a $(\text{sid}, \text{sk})$ pair should have been generated, the user $U_i$ sends $sk$ to the adversary. Otherwise, $U_i$ returns nothing.

   (g) $\text{Corrupt}_u(\text{TTP})$: The challenger returns the private information of the TTP to the adversary.

At some point, the adversary chooses $1 \leq u \leq N$ and challenges $U_u$’s $L_u^*$-th session, which has successfully ended with $(\text{sid}^*, \text{sk}^*)$. Suppose the session has been initiated to negotiate a session key with $U_u$ for an application identified by $app_id \in S$. This is subjected to the following restrictions.

   - The public keys $PK_u$ and $PK_v$ have been certified through $\text{Register}_u$ queries.
   - There has been no $\text{Corrupt}_u(\text{ID}_v)$ query and no $\text{Corrupt}_u(\text{ID}_u)$ query if $u = v$. There has been no $\text{Corrupt}_u$ query before $U_i$ and $U_j$ register their public keys.
   - There has been neither $\text{Corrupt}_u(L_u^*, \text{ID}_u)$ nor $\text{Corrupt}_v(L_v^*, \text{ID}_u)$ queries.
   - There has been neither $\text{Corrupt}_u(L_u^*, \text{ID}_u)$ nor $\text{Corrupt}_u(L_u^*, \text{ID}_v)$ query, where $U_u$’s $L_u^*$-th session possesses the session identifier $\text{sid}^*$.

3. Challenge: Select $b \in \{0, 1\}$. If $b = 0$, send $sk^*$ to the adversary, otherwise send $r \in_R K$ to the adversary where $K$ is the session key domain.

4. Phase 2: The adversary is allowed to issue the same types of queries as in Phase 1, with the following restrictions.

   - There has been neither $\text{Corrupt}_u(L_u^*, \text{ID}_u)$ nor $\text{Corrupt}_v(L_v^*, \text{ID}_u)$ queries.
   - There has been neither $\text{Corrupt}_u(L_u^*, \text{ID}_u)$ nor $\text{Corrupt}_u(L_u^*, \text{ID}_v)$ query, where $U_u$’s $L_u^*$-th session possesses the session identifier $\text{sid}^*$.

At some point, the adversary terminates by outputting a guess bit $b^*$.

Fig. 8. The Key Authentication Game

1. Setup: the challenger generates the parameters for the TTP and publishes the public parameter.

2. Challenge: The adversary is allowed to issue the following type of queries: $\text{Register}_u$, $\text{Register}_v$, $\text{Invoke}$, $\text{Corrupt}_u$, $\text{Corrupt}_v$, and $\text{Corrupt}_u$. The queries are answered in the same way as in the key authentication game, as shown in Fig. 8. At some point, the adversary chooses $1 \leq u \leq N$ and challenges $U_u$’s $L_u^*$-th session, which has successfully ended with $(\text{sid}^*, \text{sk}^*)$. Suppose the session has been initiated to negotiate a session key with $U_u$ for an application identified by $app_id \in S$. This is subjected to the following restrictions.

   - The public keys $PK_u$ and $PK_v$ have been certified through $\text{Register}_u$ queries.
   - There has been no $\text{Corrupt}_u(\text{ID}_v)$ query and no $\text{Corrupt}_u(\text{ID}_u)$ query if $u = v$. There has been no $\text{Corrupt}_u$ query before $U_i$ and $U_j$ register their public keys.

Fig. 9. The Entity Authentication Game

Definition 3. A two-party key establishment protocol achieves the entity authentication property, if any polynomial-time adversary has only negligible advantage in the entity authentication game.
5 A Secure Two-Party Key Establishment Protocol

5.1 The Proposed Protocol

We assume the users \( U_i \) (\( 1 \leq i \leq N \)) agree on a group \( G \) of prime order \( q \), a generator \( g \) of \( G \), and a hash function \( H : \{0,1\}^* \rightarrow \{0,1\}^L \) where \( L \) is a polynomial in the security parameter \( \ell \). We assume \( K = \{0,1\}^L \) is the session key domain. Every user \( U_i \) generates a key pair \( (PK_i, SK_i) \) for a digital signature scheme \((\text{KeyGen}, \text{Sign}, \text{Verify})\) which is existentially unforgeable under an adaptive chosen message attack \cite{22}, and lets the TTP certify \( PK_i \). The application identifier set is \( S = \{0,1\}^* \). If \( U_i \) and \( U_j \) want to establish a session key, they follow the protocol shown in Fig. 10.

In the protocol execution, \( U_i \) validates \( PK_j \) and \( \sigma_2 \) and aborts if any of the validations fails, and \( U_j \) validates \( PK_i \) and \( \sigma_1 \) similarly.

The proposed protocol achieves all properties defined in our security model, namely soundness, key authentication, and entity authentication. The proofs appear in Appendix B.

**Theorem 1.** The proposed protocol is sound if the hash function \( H \) is collision resistant.

**Theorem 2.** The proposed protocol achieves entity authentication property, given that the digital signature scheme \((\text{KeyGen}, \text{Sign}, \text{Verify})\) is existentially unforgeable under an adaptive chosen message attack.

**Corollary 1.** Suppose that the digital signature scheme \((\text{KeyGen}, \text{Sign}, \text{Verify})\) is existentially unforgeable under an adaptive chosen message attack. In the entity authen-
tication game, the probability that the Diffie-Hellman parameter $Y$ is not generated by $U_v$ is negligible.

Recall that the computational Diffie-Hellman (CDH) assumption holds for the group $G$ if given $g^x, g^y$ where $x, y \in_R \mathbb{Z}_q$ an adversary can only output $g^{xy}$ with a negligible probability.

**Theorem 3.** The proposed protocol achieves key authentication property based on the CDH assumption in the random oracle model, given that the digital signature scheme $(\text{KeyGen}, \text{Sign}, \text{Verify})$ is existentially unforgeable under an adaptive chosen message attack.

### 5.2 A Further Remark

In Section 3.1 we state a special case of executing a two-party key establishment protocol, i.e. a user, say $U_i$, runs the protocol between its devices while using the same long-term key pair for security guarantee. Though the proposed protocol in Section 5.1 is secure in this case, there is a higher probability that the long-term private key could be compromised since it needs to be deployed in many devices. To mitigate such risks, we can apply the following trick. Suppose that $U_i$ has two devices Laptop and Workstation, it can use the long-term key pair to certify two signature key pairs for $(PK'_i, SK'_i)$ and $(PK''_i, SK''_i)$ for the devices, respectively. Then Laptop and Workstation can use these key pairs to run the key establishment protocol.

### 6 Conclusion

In this paper, we have revisited the YAK protocol and the HMQV protocols and shown their potential weaknesses. We have presented a comprehensive model for key establishment protocols, which has taken into account the interface between a key establishment protocol and the applications which may invoke it. We have emphasized the concept of *session* and the usage of *session identifier*. Moreover, we have shown how to design a two-party key establishment protocol to achieve both key authentication and entity authentication properties in our security model. We foresee the following interesting future works. The security model can be made even stronger by allowing the adversary to access more ephemeral secrets. For example, in the key authentication game, the adversary is allowed to obtain the ephemeral secrets in section $sid^*$. The security model can be extended in other ways. For example, similar privileges can be given to the adversary as in the key-leakage resilient key establishment protocols [1]. Instead of using digital signature for authentication purpose, it is worth trying to explore public key encryption techniques. By doing so, the efficiency could be improved.
References

Appendix A. Remarks on the Security Model

The key authentication game, shown in Fig. 8, covers the extended attack scenarios listed in Section 4.2.

1. A known session key attack is covered because the adversary is allowed to obtain the available session keys (except that in the session sid*) through Corruptk queries.

2. A perfect forward secrecy attack is covered because the adversary is allowed to obtain both SKi and SKj through Corruptl queries in Phrase 2.

3. A known master key attack is covered because the adversary is allowed to obtain the TTP’s private information through a Corruptt query.

4. An unknown key-share attack is covered because the adversary is allowed to obtain the available session keys (except that in the session sid*) through Corruptk queries.

5. A key-compromise impersonation attack is covered because the adversary is allowed to obtain both SKi through Corruptt queries in Phrase 1.

In addition, the possibility of ephemeral secret leakage has been covered since the adversary is allowed to issue Corrupte queries.

The key authentication game, shown in Fig. 9, covers the extended attack scenarios listed in Section 4.3.
1. A key-compromise impersonation attack is covered because the adversary is allowed to obtain both $SK_i$ through Corrupt queries and the ephemeral secret through a Corrupt query.

2. A attack against key confirmation is covered because the adversary wins if the two users do not have the same session identifier and share the same session key.

**Appendix B. Proof of Theorems**

*Proof sketch of Theorem 1.* Briefly, we show that the three requirements in Definition are satisfied.

1. It is straightforward to verify that, at the end of a session, the two users obtain the same $(sid, sk)$ pair.

2. For $U_i$, the session identifier is computed partially with the input $g^x$, where $x$ is randomly chosen in every session. Clearly, given that the hash function $H$ is collision resistant, two sessions of $U_i$, carried out with $U_j$ for an application $appid$, the probability that the session identifiers are identical is negligible.

3. For a session between $U_i$ and $U_j$ for the application $appid$, the session identifier is computed partially with the input

$$PK_i||ID_i||PK_j||ID_j||appid.$$ 

For a session between $U_k$ and $U_l$ for the application $appid'$, the session identifier is computed partially with the input

$$PK_k||ID_k||PK_l||ID_l||appid'.$$

Clearly, given that the sets $\{ID_i, ID_j, appid\}$ and $\{ID_k, ID_l, appid'\}$ are not identical and the hash function $H$ is collision resistant, the probability that the session identifier of $ID_i$ and the session identifier of $ID_k$ are identical is negligible, regardless of the coin flips of all users.

The theorem now follows.

*Proof sketch Theorem 2.* Suppose that $U_u$ successfully ended with a session with $U_v$ for the application $appid$. Suppose also that $U_u$ obtains $(sid^*, sk^*)$ at the end of the session. Based on the protocol specification, $U_u$ has received a valid signature of the form

$$\sigma_2 = \text{Sign}(1||W||H(Y^{log_x} SK_v), SK_v),$$

where

$$sid^* = H(W)$$

and

$$W = PK_u||ID_u||PK_v||ID_v||appid||X||Y.$$ 

Now, suppose the user $U_u$ does not have a session with the session identifier $sid^*$ and session key $sk^* = H(W||X^{log_x})$, then the probability that $U_u$ has generated a signature equalling to $\sigma_2$ is 0. Note that the adversary is not allowed to compromise $SK_v$ in the game. As a result, the adversary has forged the signature $\sigma_2$, which has a negligible probability since the digital signature scheme $(\text{KeyGen}, \text{Sign}, \text{Verify})$ is
existentially unforgeable under an adaptive chosen message attack. The theorem now follows.

Proof sketch Theorem 3. Suppose that there are at most \( T \) Invoke oracle queries in Phase 1 of the key authentication game. Since we only consider a polynomial-time adversary, \( T \) is a polynomial in the security parameter \( \ell \). Let’s the adversary’s advantage be \( \epsilon \) in the key authentication game, shown in Fig. 8. The rest of the security proof is done through a sequence of games [26].

**Game**\(_0\): At the beginning of this game, the challenger makes a prediction on the session that the adversary would challenge in the game. The guess is correct implies that it is a \( U_u^{\ast} \)'s \( L_u^{\ast} \)-th session with \( U_{\ast} \) for an application \( app_{\ast id} \). If the guess is incorrect, the challenger aborts. Otherwise, the challenger faithfully simulates the protocol execution and answers the oracle queries from the adversary \( A \). Let \( \epsilon_1 \) be the probability that the challenger does not abort and the adversary wins the game. Clearly, the probability that the challenger does not abort is \( \frac{1}{T} \), in which case the adversary’s advantage is \( \epsilon \) (in this case, **Game**\(_0\) is equivalently the key authentication game). Therefore, \( \epsilon_1 = \frac{\epsilon}{T} \).

**Game**\(_1\): In this game, the challenger perform in the same way as in **Game**\(_0\) except the session key is computed as

\[
sid^* = H(W), \ sk^* = R, \ R \in \{0,1\}^L, \ W = PK_u || ID_u || PK_v || ID_v || app_{\ast id} || X || Y.
\]

Let \( \epsilon_2 \) be the probability that the challenger does not abort the game and the adversary wins the game. Clearly, the probability that the challenger does not abort is \( \frac{1}{T} \), in which case the adversary’s advantage is exactly 0 since the challenger returns a random message, regardless the coin flip \( b \). Therefore, \( \epsilon_2 = \frac{0}{T} \).

If the hash function is modeled as a random oracle, then **Game**\(_1\) is identical to **Game**\(_0\) unless the adversary has queries the random oracle \( H \) with the input \( W || Y^{log_X} \), which has a negligible probability based on Corollary 1 and the CDH assumption. We have \( |\epsilon_2 - \epsilon_1| \), namely \( \frac{\epsilon}{T} \), is negligible from the Difference Lemma in [26]. Since \( T \) is a polynomial in the security parameter \( \ell \), the value \( \epsilon \) is negligible. The theorem now follows. \( \square \)