Improving the Security of the HMQV Protocol Using Tamper-Proof Hardware

Qianying Zhang(✉), Shijun Zhao, Yu Qin, and Dengguo Feng

Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
zsjzqy@gmail.com, {zhaosj,qin_yu,feng}@tca.iscas.ac.cn

Abstract. The full Perfect Forward Secrecy (PFS) is an important security property for Authenticated Key Exchange (AKE) protocols. Unfortunately, Krawczyk has claimed that any one-round implicitly authenticated key exchange protocol could not achieve full PFS but only weak PFS. Although some solutions are proposed in the literature, their protocols maintain secure only in the cases of additional authentication and a constrained adversary. In this paper, we investigate the question of whether tamper-proof hardware can circumvent the full PFS deficiency of one-round implicitly authenticated key exchange protocols. We answer this question in the affirmative by formally proving that the most efficient one-round implicitly authenticated key exchange protocol, HMQV, achieves full PFS under the physical assumption of regarding the existence of tamper-proof hardware.

Keywords: Authenticated Key Exchange · Full PFS · Tamper-Proof hardware · Physical assumption · HMQV · CK model

1 Introduction

Diffie and Hellman gave the first key exchange protocol in their seminal paper [9]. Key exchange protocols allow two entities to establish a shared secret session key via public communication. In order to provide the authentication of entities' identities, authenticated key exchange (AKE) was proposed. AKE not only allows two entities to compute a shared session key but also ensures the authenticity of the entities. In this paper, we focus on a kind of AKE protocol put forth by Matsumoto [23] which needs only the basic Diffie-Hellman exchanges, yet it provides authentication by combining the ephemeral keys and long-term keys in the derivation of the session key. As this kind of protocol achieves high performance both in communication (only the basic Diffie-Hellman exchanges are needed) and computation (needs no explicit signature authentication), it is widely studied and many protocols are proposed [16,19–22,25,29,32–34].

The full PFS is a desirable property for AKE protocols. It ensures that the expired session keys established before the compromise of the long-term key cannot be recovered even if the adversary is active during the session establishment. However, Krawczyk showed in his well-known protocol HMQV [19]

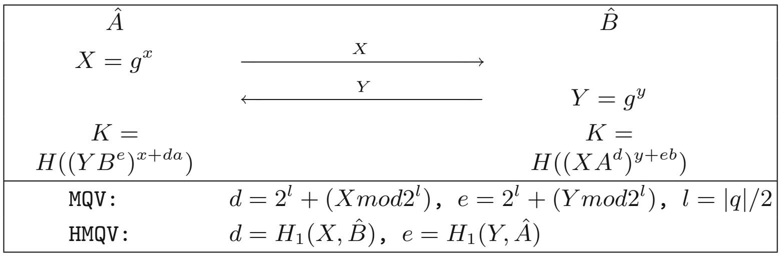
J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 343–361, 2015.
DOI: 10.1007/978-3-319-23829-6_24

that any one-round (or two-message) implicitly authenticated key exchange protocol could not achieve the full PFS property, and gave an explicit attack on such protocols. He claimed that implicitly authenticated protocols could only achieve weak PFS: "any session key established without the active intervention of the attacker (except for eavesdropping the communication) is guaranteed to be irrecoverable by the attacker once the session key is erased from memory". Boyd and Gonzalez [2] further proved that if the adversary is allowed to reveal the ephemeral keys then no one-round AKE protocols can achieve full PFS. In the following we show the reason why one-round implicitly authenticated key exchange protocols cannot achieve full PFS by analyzing the HMQV [19] protocol.

HMQV originates from the MQV protocol [22], and is one of the most efficient one-round implicitly authenticated key exchange protocols. It achieves almost the strongest security requirements for AKE, i.e., provable security in the CK model, resistance to the key-compromise impersonation attacks and weak PFS property. Krawczyk formally proves its security in the CK model [3]. However, in scenarios where the ephemeral keys are not protected, the validation of the ephemeral public key must be performed explicitly, which costs one exponentiation, or the protocol would be vulnerable to small subgroup attacks [24].

The HMQV protocol is depicted in Fig. 1. It involves two entities $\hat{A}$ and $\hat{B}$, with respective secret keys a and b and public keys $A = g^a$ and $B = g^b$. First, entities $\hat{A}$ and $\hat{B}$ randomly select ephemeral private keys x and y and exchange the ephemeral public keys X and Y. Then both entities compute a session key K as $H(g^{(x+da)(y+eb)})$ where $d = H_1(X, \hat{B})$, $e = H_1(Y, \hat{A})$ and H, H_1 are hash functions.

We review the attack on full PFS of HMQV in the following. An adversary $\mathcal{M}$ randomly chooses a secret key x and sends the public key $X = g^x$ to $\hat{B}$ masquerading as $\hat{A}$. Then $\hat{B}$ will choose a random secret key y, send $Y = g^y$ to $\hat{A}$ which is captured by $\mathcal{M}$, and compute the session key $K = H((XA^d)^{y+eb})$. Once the session key expires at $\hat{B}$, and is removed from memory, $\mathcal{M}$ corrupts $\hat{A}$ and obtains the private key a. $\mathcal{M}$ now can compute the session key K by computing $H((YB^e)^{x+da})$ which contradicts the full PFS property. The above attack on HMQV can be easily applied to all the one-round implicitly authenticated key exchange protocols. So it seems impossible to achieve full PFS for such protocols.



$\hat{A}$
$\hat{B}$
$X = g^x$
X
Y
$Y = g^y$
$K = H((YB^e)^{x+da})$
$K = H((XA^d)^{y+eb})$
MQV: $d = 2^l + (X mod 2^l)$, $e = 2^l + (Y mod 2^l)$, $l = |q|/2$
HMQV: $d = H_1(X, \hat{B})$, $e = H_1(Y, \hat{A})$

Fig. 1. The MQV and HMQV protocol

1.1 Related Work and Contributions

Related Work. The tamper-proof hardware token stores sensitive data such as cryptographic keys and protected objects in its shielded memory and provides users of the token its cryptographic functionalities through secure API such as PKCS#11 [28]. The tamper-proof feature of the token and the secure API protect sensitive data in the hardware from being revealed in plaintext off the token. Moreover, tamper-proof hardware associates every protected object with an authorization, and only users possessing the correct authorization can make use of the functionality of the object. So even if the adversary corrupts some object by compromising the authorization, it only gets the black-box access to the object through the secure API but not the plaintext.

The idea of using secure hardware to achieve stronger security properties is not entirely new, and a number of works based on tamper-proof hardware have been proposed. Katz [17] first formalizes tamper-proof hardware in the universal composability (UC) framework and proves that such physical assumptions suffice to circumvent the impossibility result of secure computation of general functionalities without an honest majority. Some following papers [4,8,26] give further investigation. Goldwasser et al. [11] introduce the concept of one-time programs, in which they make use of very simple hardware tokens to ensure that a program is used only once. Goyal et al. [13] consider the general question of basing secure computation on hardware tokens, and show some impossible cryptographic tasks in the “plain” model become feasible if the entities are allowed to generate and exchange tamper-proof hardware tokens. Dagdelen et al. [7] present an efficient protocol for password-based authenticated key exchange based on the weak model of one-time memory tokens [11]. Kolesnikov [18] proposes a truly efficient String Oblivious Transfer (OT) technique relying on resettable (actually, stateless) tamper-proof tokens. [12,15] focus on the possibilities of efficient Zero-Knowledge PCPs and unconditional two-prover Zero-Knowledge proofs for **NP** on stateless tamper-proof hardware tokens respectively.

Our Contributions. In this paper we extend the idea of improving the security of cryptography protocols using tamper-proof hardware to modern AKE protocols. We first design the API of tamper-proof hardware for the HMQV protocol, then in our formal analysis we model the black-box manner of the tamper-proof hardware API as an oracle, i.e., instead of getting the plaintext of the private key, the adversary gets an API oracle after compromising the long-term key. Under the assumption of the existence of tamper-proof hardware, we formally prove that the HMQV protocol achieves the full PFS property in the CK model. Although it seems a bit trivial by using a tamper-proof hardware to achieve full PFS. Evidently it is not such a trivial task and a challenging work, given the state-of-the-art nature and highly intensive study of HMQV.

Another advantage of our design of the tamper-proof hardware API is that our HMQV design can resist small subgroup attacks even if entities don’t perform the validation of ephemeral public keys. So the total computation cost of our HMQV per entity is only 2.5 exponentiations.

1.2 Organization

Section 2 gives a brief description of the CK model. Section 3 summarizes the current one-round AKE protocols achieving full PFS, presents their limitations, and gives a detailed comparison with our HMQV protocol. Section 4 designs the API of tamper-proof hardware for HMQV, explains why our design resists small group attacks even if ephemeral public keys are not validated, and gives a formal description of HMQV. Section 5 formally proves the security of HMQV in the CK model and shows that it achieves full PFS with the help of tamper-proof hardware. Section 6 concludes our work and gives our future work.

2 Security Model for AKE

We outline the CK model for key exchange protocols on which all the analysis work in this paper is based. In the CK model, AKE runs in a network of interconnected entities and each entity has a long-term key and a certificate (issued by a certification authority (CA)) that binds the public key with the identity of that entity. An entity can be activated to run an instance of the protocol called a session. Within a session an entity can be activated to initiate the session or to respond to an incoming message. As a result of these activations, the entity creates and maintains a session state, generates outgoing messages, and eventually completes the session by outputting a session key and erasing the session state. A session can be associated with its holder or owner (the entity at which the session exists), a peer (the entity with which the session key is intended to be established), and a session identifier. The session identifier is a 4-tuple $(\hat{A}, \hat{B}, out, in)$ where $\hat{A}$ is the identity of the owner of the session, $\hat{B}$ the peer, out the outgoing messages from $\hat{A}$ in the session, and in the incoming messages from $\hat{B}$. In the case of the one-round implicitly authenticated key exchange protocols, this results in an identifier of the form $(\hat{A}, \hat{B}, X, Y)$ where X is the outgoing DH value and Y the incoming DH value. The session $(\hat{B}, \hat{A}, Y, X)$ (if it exists) is said to be **matching** to session $(\hat{A}, \hat{B}, X, Y)$.

2.1 Attack Model

The AKE experiment involves multiple honest entities and an adversary $\mathcal{M}$ connected via an unauthenticated network. The adversary is modeled as a probabilistic Turing machine and has full control of the communications between entities. $\mathcal{M}$ can intercept and modify messages sent over the network. $\mathcal{M}$ also schedules all session activations and session-message delivery. In addition, in order to model potential disclosure of secret information, the adversary is allowed to access secret information via the following queries:

- **SessionStateReveal(s):** $\mathcal{M}$ queries directly at session s while still incomplete and learns the session state for s. This query allows the adversary to obtain all states stored on the untrusted host, such as the values returned by the API of tamper-proof hardware and all the information computed on the host.

– **SessionKeyReveal(s):** $\mathcal{M}$ obtains the session key for the session s.
– **Corruption($\hat{P}$):** In the "plain" CK model (In this paper we use the term "plain model" to denote the model that has no tamper-proof hardware assumption), this query allows $\mathcal{M}$ to learn the plaintext of the long-term private key of entity $\hat{P}$. In the tamper-proof hardware model, $\mathcal{M}$ cannot learn anything about the plaintext of the private key but gets the black-box access to the private key as the hardware is completely tamper-proof. In other words, this query allows $\mathcal{M}$ to obtain an API oracle of the private key.
– **Expiry(s):** This query deletes the session key and any related session state of session s. While it has no output, expiry is of major importance in defining full PFS.
– **Test(s):** Pick $b \overset{R}{\leftarrow} 0, 1$. If $b = 1$, provide $\mathcal{M}$ the session key; otherwise provide $\mathcal{M}$ with a value r randomly chosen from the probability distribution of session keys. This query can only be issued to a session that is "clean". We say that a completed session is "clean" if this session as well as its matching session (if it exists) is not subject to any of the first 3 queries above (SessionStateReveal, SessionKeyReveal, Corruption). A session is called *exposed* if $\mathcal{M}$ performs any one of the first 3 queries to this session.

The security is defined based on a game played by $\mathcal{M}$, in which $\mathcal{M}$ is allowed to activate sessions and perform Corruption, SessionStateReveal, SessionKeyReveal and Expiry queries. At some time, $\mathcal{M}$ performs the Test query to a clean session of its choice and gets the value returned by Test. After that, $\mathcal{M}$ continues the experiment, but is not allowed to expose the test session nor any entities involved in the test session. However, in order to model full PFS we allow the adversary to corrupt the owner of the test session and the peer entity after the session has expired. Eventually $\mathcal{M}$ outputs a bit b' as its guess, then halts. $\mathcal{M}$ wins the game if $b' = b$. The adversary with above capabilities is called a **KE-adversary**. We give the formal definition of security in the following.

Definition 1. *An AKE protocol Π is called secure if the following properties hold for any KE-adversary $\mathcal{M}$ defined above:*

1. *When two uncorrupted entities complete matching sessions, they output the same session key, and*
2. *The probability that $\mathcal{M}$ guesses the bit b (i.e., outputs $b' = b$) from the Test query correctly is no more than 1/2 plus a negligible fraction.*

3 Current Limitations and Comparisons

In this section, we summarize all the one-round AKE protocols achieving full PFS as far as we know, and present their limitations. At last we compare these protocols with our HMQV protocol with hardware assumption.

3.1 Current AKE Achieving Full PFS and Their Limitations

Many one-round protocols with full PFS [2,5,6,10,14,16,35] have been proposed especially after Krawczyk pointed out the full PFS deficiency of one-round protocols, although many of them are not implicitly authenticated as they need to

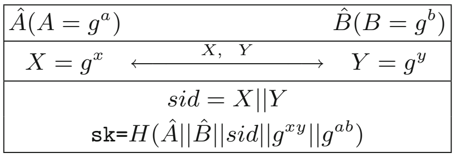


Fig. 2. The $\mathcal{TS}2$ protocol

explicitly authenticate the transmitted messages to prevent the adversary from injecting self-constructed messages.

The protocol $\mathcal{TS}2$ of Jeong, Katz and Lee [16] and the mOT protocol of Gennaro et al. [10] are typical efficient one-round authenticated key exchange protocols. Take $\mathcal{TS}2$ for example, any two parties willing to establish a shared session key between them first exchange their ephemeral values, and then derive the session key by combing the ephemeral values and the long-term keys. Figure 2 illustrates the generic protocol messages and session key computation of $\mathcal{TS}2$. However, models used in the security proofs of $\mathcal{TS}2$ and mOT do not allow any ephemeral values and intermediate information to be revealed. We can see that if the adversary is allowed to reveal the ephemeral keys then $\mathcal{TS}2$ can't achieve full PFS as Boyd and Gonzalez have analyzed in [2]. What's worse, we find that if the adversary is allowed to reveal the intermediate information g^{ab} then the $\mathcal{TS}2$ protocol is completely insecure: the adversary transmits an ephemeral key $X' = g^{x'}$ generated by himself to entity $\hat{A}$ or $\hat{B}$ and then computes the session key $K' = H(\hat{A}||\hat{B}||sid||Y^{x'}||g^{ab})$, in another word, the adversary is able to impersonate $\hat{A}$ (or $\hat{B}$) to $\hat{B}$ (or $\hat{A}$) indefinitely. Authors of the mOT protocol [10] point out that mOT "is not resistant to the disclosure of the ephemeral Diffie-Hellman values or the unhashed session key": if the adversary is allowed to reveal the ephemeral keys the adversary can immediately obtain the sender's long-term private key, and if the adversary is allowed to reveal the unhashed session key the adversary can carry a malleability attack. As no state information is allowed to disclose, the security models used in $\mathcal{TS}2$ and mOT are similar to the Bellare-Rogaway model [1], which is weaker than the popular CK or eCK model. At a practical level, the ephemeral key must be protected with the same security as the long-term private keys and all the intermediate computation must be performed in tamper-proof device. Thus, such protocols are not efficient for tamper-proof hardware whose physical resources might be very limited.

After the proposal of $\mathcal{TS}2$ and mOT protocols, many one-round protocols with full PFS and proved secure in strong models (such as CK and eCK models) are proposed [2,5,6,14,35]. Here we only give a detail analysis of the limitations of the protocol presented by Cremers [6][1], and our analysis can be applied to other protocols easily as their mechanisms used to provide full PFS are similar.

[1] Actually [2,6] give compilers that transmit a one-round protocol into a protocol with full PFS property, and here we analyze the transformations of the NAXOS protocol which are presented as typical examples in their papers.

Protocol [6] is a variant of the NAXOS protocol [20], and is showed in Fig. 3. The first limitation is that the security model used in the analysis disallows the adversary to reveal the ephemeral keys of all the sessions whose output messages are the same as the input messages of the test session, i.e. the session under attack. Such limitation exists in the models of other protocols: [2] disallows the adversary to get any state information of the peer to the test session, and [14] disallows the adversary of the eCK model to reveal any ephemeral keys, and [35] disallows the adversary to reveal any session state information between the owner and the peer of the test session. These constraints indeed help the above protocols achieve full PFS in their security models, but they prevent their models from capturing the attacks launched by a "clever" active adversary who would always replay messages of such sessions whose ephemeral keys or state information he has already obtained, i.e., such adversary would first corrupt the ephemeral key or the session state of some session and then replay the corrupted messages to some entities. The second limitation of protocol [6] is that each entity needs to authenticate the exchanged messages (such as the signatures in Fig. 3) using an extra signature key (such as $(sk_{\hat{A}}, pk_{\hat{A}})$ of $\hat{A}$ in Fig. 3), which adds a signature computation to each entity. Boyd and Gonzalez claimed [2] is more efficient as they use a MAC instead of a signature, but the computation of the secret key used in the MAC costs an exponentiation. Huang's protocol [14] doesn't use an additional signature key, but the authentication of the ephemeral key is performed by the long-term key and it doesn't work over arbitrary groups as it requires a decisional Diffie-Helleman (DDH) oracle.

$$\begin{array}{|lcr|} \hline \hat{A} & & \hat{B} \\ (a,A),(sk_{\hat{A}},pk_{\hat{A}}) & & (b,B),(sk_{\hat{B}},pk_{\hat{B}}) \\ \hline r_{\hat{A}}, X = g^{H_1(r_{\hat{A}},a)} & \begin{array}{c} \xrightarrow{X,Sign_{sk_{\hat{A}}}(X[,\hat{B}])} \\ \xleftarrow{Y,Sign_{sk_{\hat{B}}}(Y[,\hat{A}])} \end{array} & r_{\hat{B}}, Y = g^{H_1(r_{\hat{B}},b)} \\ \hline \multicolumn{3}{|c|}{K_{\hat{A}} = H_2(Y^a, B^{H_1(r_{\hat{A}},a)}, Y^{H_1(r_{\hat{A}},a)}, \hat{A}, \hat{B})} \\ \multicolumn{3}{|c|}{K_{\hat{B}} = H_2(A^{H_1(r_{\hat{B}},b)}, X^b, X^{H_1(r_{\hat{B}},b)}, \hat{A}, \hat{B})} \\ \hline \end{array}$$

Fig. 3. The variant of NAXOS

From above we conclude that current solutions on the full PFS deficiency of one-round AKE protocols are not perfect: they maintain protocol security and full PFS only in weak security models or in strong models while the capabilities of the adversary is constrained and the exchanged messages are explicitly authenticated by signature or MAC.

3.2 Comparisons

Our HMQV protocol with hardware assumption only disallows the adversary to reveal the sensitive information stored and computed in the tamper-proof

Table 1. Protocol Comparisons

Protocol	Efficiency	Validation	Communication	Assumption
$\mathcal{TS}2$ [16]	3	Y	2|P|	CDH, RO
$\mathcal{TS}3$ [16]	3	Y	2|P| + 2 |MAC|	CDH
mOT [10]	2	Y	2|N|	RSA, KEA1, RO
Boyd11 [2]	5	Y	2|P| + 2 |MAC|	GDH, RO
Cremers11 [5]	3 + 1 Sign	Y	2|P| + 2 |Sign|	GDH, RO
Cremers12 [6]	4 + 1 Sign	Y	2|P| + 2 |Sign|	GDH, RO
Huang11 [14]	3 + 1 DDH	Y	4|P|	GDH, RO
Yoneyama12 [35]	8 + 2 Pair	N	10|P|	DDH, DBDH, q-SDH
Our HMQV	2.5	N	2|P|	GDH, Physical, RO

In the Efficiency column, the numbers denote the exponentiations, and Sign denotes the computation of a signature, and DDH denotes the computation of querying a DDH oracle, and Pair denotes the paring computation. In the Communication column, |P| denotes the size of a group element, and |N| denotes the size of an RSA key, and |MAC| denotes the size of a MAC, and |Sign| denotes the size of a signature. The CDH, RSA, KEA1, DDH, GDH, DBDH and q-SDH stand for the Computational Diffie-Hellman, RSA, Knowledge of Exponent, Decisional Diffie-Hellman, Gap-DDH, Decisional Bilinear Diffie-Hellman, q-strong Diffie-Hellman assumptions respectively, and RO is short for the random oracle model.

hardware, and other information such as results returned by the hardware API, is allowed to be revealed to the adversary in our security analysis. In Table 1 we compare our HMQV with other one-round AKE protocols achieving full PFS in terms of the efficiency, necessity of validation of the ephemeral public keys, communication, and the underlying hardness assumptions. Table 1 shows that our HMQV protocol is almost the most efficient both in computation and communication (except for the mOT protocol in the efficiency, but mOT only works for RSA groups whose exponentiation computation is more expensive).

4 API Design and Protocol Description

In this section, we introduce the design of tamper-proof hardware API for the HMQV protocol, explain why no adversaries can mount small group attacks even if the validation of ephemeral public keys is eliminated, and then give a formal description of the protocol.

4.1 API Design and the Resistance to Small Group Attacks

Tamper-proof hardware stores the long-term private key of its owner, and provides its owner two functionalities through the API: (1) generating an ephemeral key, and (2) generating the unhashed shared secret based on the long-term keys and the ephemeral keys. Figure 4 depicts the API, and we now give a detailed description:

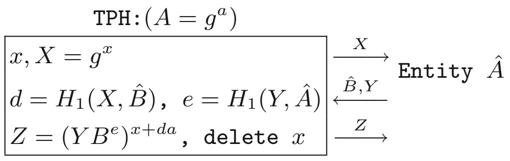

TPH: $(A = g^a)$
$x, X = g^x$
$d = H_1(X, \hat{B})$, $e = H_1(Y, \hat{A})$
$Z = (YB^e)^{x+da}$, delete x
X
$\hat{B}, Y$
Z
Entity $\hat{A}$

Fig. 4. The API of Tamper-Proof hardware

1. When entity $\hat{A}$ wishes to establish a session key with entity $\hat{B}$, it first calls the API of its tamper-proof hardware to get an ephemeral public key $X = g^x$. The ephemeral private key x is stored in the hardware, and the public key X will be used to exchange with $\hat{B}$.
2. After receiving the ephemeral key Y from entity $\hat{B}$, $\hat{A}$ transmits $(\hat{B}, Y)$ to its tamper-proof hardware through the API, and the hardware will perform the following steps:
   (a) Compute $d = H_1(X, \hat{B})$ and $e = H_1(Y, \hat{A})$ where H_1 is a hash function. d and e are of length $|p|/2$ where $|p|$ is the bit length of the group order.
   (b) Compute the unhashed shared secret $Z = (YB^e)^{x+da}$, delete x, then return Z to $\hat{A}$.

After receiving Z from its tamper-proof hardware, $\hat{A}$ can compute the session key shared with $\hat{B}$ by hashing Z. The details will be introduced in Sect. 4.2.

Resistance to Small Group Attacks. As Menezes [24] has shown that if the ephemeral private key is allowed to be exposed to the adversary in the session state query, key exchange protocols are vulnerable to small subgroup attacks, which allow the adversary to recover long-term private keys. For the details of small subgroup attacks, please refer to [24]. So in our design of tamper-proof hardware API for HMQV, ephemeral keys are generated by the tamper-proof hardware and ephemeral private keys are physically protected. As the generation of ephemeral keys doesn't require any information of the peer entity, ephemeral keys can be generated off-line (when tamper-proof hardware is ideal). Thus, putting the generation of ephemeral keys into tamper-proof hardware doesn't affect the efficiency of HMQV in practice. To demonstrate that our design is practical, we study ephemeral key generation of the Trusted Platform Module (TPM) version 2.0 [30] (although TPM 2.0 is not a tamper-proof hardware, it is a popular hardware security token). We find that TPM 2.0 designs an efficient way to generate ephemeral keys with the following features:

– have the number of bits equal to the security strength of the signing key;
– not be known outside of the TPM; and
– only be used once.

Users can invoke the TPM2_EC_Ephemeral() [31] command to generate an ephemeral key. So we claim that protecting ephemeral private keys by tamper-proof hardware is practical.

4.2 Formal Description of HMQV

Figure 5 gives an informal description of HMQV, and the computation performed by tamper-proof hardware is boxed by rectangles. We formally describe HMQV by giving the following three session activations.

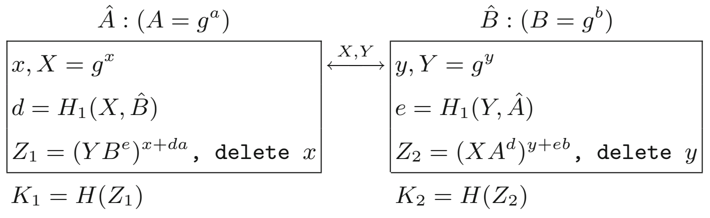


Fig. 5. The HMQV protocol

1. Initiate($\hat{A}, \hat{B}$): $\hat{A}$ calls the API of its tamper-proof hardware to generate an ephemeral key X, creates a local session of the protocol which it identifies as (the incomplete) session $(\hat{A}, \hat{B}, X)$, and outputs X as its outgoing message.
2. Respond($\hat{B}, \hat{A}, X$): After receiving X, $\hat{B}$ performs the following steps:
   (a) Call the API of its tamper-proof hardware to get an ephemeral key Y, output Y as its outgoing message.
   (b) Transmit $(\hat{A}, X)$ to its tamper-proof hardware and get $Z_2 = (XA^d)^{y+eb}$ through the API where y is the private part of Y and $d = H_1(X, \hat{B})$, $e = H_1(Y, \hat{A})$.
   (c) Compute the session key $K_2 = H(Z_2)$ and complete the session with identifier $(\hat{B}, \hat{A}, Y, X)$.
3. Complete($\hat{A}, \hat{B}, X, Y$): $\hat{A}$ checks that it has an open session with identifier $(\hat{A}, \hat{B}, X)$, then performs the following steps:
   (a) Transmit $(\hat{B}, Y)$ to its tamper-proof hardware and get $Z_1 = (YB^e)^{x+da}$ through the API where x is the private part of X and $d = H_1(X, \hat{B})$, $e = H_1(Y, \hat{A})$.
   (b) Compute the session key $K_1 = H(Z_1)$ and complete the session with identifier $(\hat{A}, \hat{B}, X, Y)$.

It is straightforward to verify that the two entities compute the same shared secret $Z = Z_1 = Z_2$ and the same session key $K = K_1 = K_2$.

5 Security Proof of HMQV with Tamper-Proof Hardware

We first describe the GDH (Gap Diffie-Hellman) assumption, then prove our HMQV protocol is secure and achieves the full PFS property in the CK model under the GDH assumption.

Definition 2 (GDH Assumption). *Let G be a cyclic group generated by an element g whose order is p. We say that a decision algorithm $\mathcal{DDH}$ is a Decisional Diffie-Hellman (DDH) Oracle for a group G and generator g if on input a triple (X, Y, Z), for $X, Y \in G$, oracle $\mathcal{DDH}$ outputs 1 if and only if Z=CDH(X, Y). We say that G satisfies the GDH assumption if no feasible algorithm exists to solve the CDH problem, even when the algorithm is provided with a DDH-oracle for G.*

API Oracle. We treat the API of tamper-proof hardware as an oracle $\mathcal{O}$ who generates ephemeral keys and unhashed shared secrets. The adversary would be given the black-box access to $\mathcal{O}$ if it performs the Corruption query to the entity.

Session State. In order to simulate the tamper-proof feature of the hardware, we specify that a session state stores the results returned by the API, i.e., the unhashed shared secret Z. Information stored in the hardware is not included in the session state, for example, ephemeral private keys.

Theorem 1. *Under the GDH assumption, the HMQV protocol, with hash functions H and H_1 modeled as random oracles, is a secure key exchange protocol in the CK model described in Sect. 2.*

The proof of the above theorem follows from the definition of secure key exchange protocols outlined in Sect. 2 and the following two lemmas.

Lemma 1. *If two entities $\hat{A}$, $\hat{B}$ complete matching sessions, then their session keys are the same.*

Lemma 2. *Under the GDH assumption, there is no feasible adversary that succeeds in distinguishing the session key of an unexposed session with non-negligible probability.*

Lemma 1 follows immediately from the definition of matching sessions. That is, if $\hat{A}$ completes session $(\hat{A}, \hat{B}, X, Y)$ and $\hat{B}$ completes the matching session $(\hat{B}, \hat{A}, Y, X)$ then $\hat{A}$ computes its session key as $H(Z_1)$ while $\hat{B}$ computes the same key as $H(Z_2)$ where $Z_1 = Z_2$.

The rest section proves Lemma 2. Let $\mathcal{M}$ be any adversary against our HMQV protocol. We observe that since the session key of the test session is computed as $K = H(Z)$ for Z, the adversary $\mathcal{M}$ has only two ways to distinguish K from a random value:

1. Forging attack. At some point $\mathcal{M}$ queries H on the same Z as the unhashed shared secret of the test session.
2. Key-replication attack. $\mathcal{M}$ succeeds in forcing the establishment of another session that has the same session key as the test session.

For simplicity of analysis we will consider the above two forms of attacks separately. We will show that if either of the attacks succeeds with non-negligible probability then there exists an efficient solver $\mathcal{S}$ against the GDH problem.

5.1 Infeasibility of Forging Attacks

Consider a successful run of $\mathcal{M}$, and let $(\hat{A}, \hat{B}, X_0, Y_0)$ denote the test session for which $\mathcal{M}$ outputs a correct guess for the Z value of the test session. By the convention on session identifiers, we know that the test session is held by $\hat{A}$, and its peer is $\hat{B}$, X_0 was output by $\hat{A}$, and Y_0 was the incoming message to $\hat{A}$. The generation of the Y_0 can fall under one of the following three cases:

1. Y_0 was generated by $\hat{B}$ in a session matching the test session, i.e., in session $(\hat{B}, \hat{A}, Y_0, X_0)$.
2. Y_0 was never output by $\hat{B}$ as its outgoing value in any of the sessions activated at $\hat{B}$, or $\hat{B}$ did output Y_0 as its outgoing value for some session s but it never completed the session key of s ($\hat{B}$ was invoked to execute the Initiate activation of s but was never activated with Complete activation).
3. Y_0 was generated at $\hat{B}$ during a non-matching session $(\hat{B}, \hat{A}^*, Y_0, X^*)$ with $\hat{A}^* \neq \hat{A}$ or $X^* \neq X_0$.

Since we assume that $\mathcal{M}$ succeeds in the forging attack with non-negligible probability then there at least one of the cases that happens with non-negligible probability in the successful run of $\mathcal{M}$. For each of the cases we build a solver $\mathcal{S}$ against the GDH problem. We assume that $\mathcal{M}$ operates in an environment that involves at most n entities and each entity participates in at most k sessions.

Solver $\mathcal{S}$ for case 1. In this case $\mathcal{S}$ takes as input a pair $(X_0, Y_0) \in G^2$, creates an AKE experiment which includes n entities, and is given access to a DDH oracle $\mathcal{DDH}$. $\mathcal{S}$ assigns the n entities random static key pairs, then randomly selects two integers $i, j \in [1, ..., k]$ and two honest entities $\hat{A}$ and $\hat{B}$. $\mathcal{S}$ runs HMQV under the control of $\mathcal{M}$ who schedules all session activations and makes queries as follows:

1. Initiate$(\hat{P}_1, \hat{P}_2)$: $\hat{P}_1$ executes the Initiate() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}$ checks whether $\hat{P}_2$ is $\hat{B}$ (or $\hat{A}$). If so, $\mathcal{S}$ sets the ephemeral public key to be X_0 (or Y_0) from the input of $\mathcal{S}$. Otherwise, $\mathcal{S}$ aborts.
2. Respond$(\hat{P}_1, \hat{P}_2, Y)$: $\hat{P}_1$ executes the Respond() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}$ checks whether $Y = Y_0$ (or $Y = X_0$). If so, $\mathcal{S}$ sets the ephemeral public key to be X_0 (or Y_0), and completes the session without computing a session key. Otherwise, $\mathcal{S}$ aborts.
3. Complete$(\hat{P}_1, \hat{P}_2, X, Y)$: $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}$ checks whether it has an open session with identifier $(\hat{A}, \hat{B}, X_0)$ (or $(\hat{B}, \hat{A}, Y_0)$) and $Y = Y_0$ (or $Y = X_0$). If so, $\mathcal{S}$ completes the session without computing a session key. Otherwise, $\mathcal{S}$ aborts.
4. SessionStateReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the unhashed shared secrete Z. However, if s is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}$ aborts.
5. SessionKeyReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}$ aborts.

6. Corruption($\hat{P}$): $\mathcal{S}$ gives $\mathcal{M}$ the API oracle $\mathcal{O}_{\hat{P}}$ of $\hat{P}$ and state information for current sessions and session keys at $\hat{P}$. From the moment of corruption $\mathcal{M}$ takes full control over $\hat{P}$ with the help of $\mathcal{O}_{\hat{P}}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ (or $\hat{B}$) when the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$) is not expired, $\mathcal{S}$ aborts.
7. Expiry(s): $\mathcal{S}$ deletes the session key and any related session state of s.
8. $H_1(\cdot)$: $\mathcal{S}$ simulates a random oracle in the usual way.
9. $H(Z)$ for some Z proceeds as follows:
   - If $\mathcal{DDH}(X_0A^d, Y_0B^e) = 1$ for Z where $d = H_1(X_0, \hat{B})$ and $e = H_1(Y_0, \hat{A})$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting:
$$\text{CDH}(X_0, Y_0) = ZX_0^{-eb}Y_0^{-da}g^{-deab}.$$
   - $\mathcal{S}$ simulates a random oracle in the usual way.

Proof. The probability that $\mathcal{M}$ selects the i-th session of $\hat{A}$ and the j-th session of $\hat{B}$ as the test session and its matching session is at least $\frac{2}{(nk)^2}$. Suppose that this is indeed the case, $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ before its i-th session is expired and $\hat{B}$ before its j-th session is expired, make SessionStateReveal and SessionKeyReveal queries to the two special sessions, so $\mathcal{S}$ doesn't abort in Step 1, 2, 3, 4, 5, 6. So $\mathcal{S}$ perfectly simulates $\mathcal{M}$'s environment except with negligible probability. Therefore if $\mathcal{M}$ wins the forging attack, then the success probability of $\mathcal{S}$ is bounded by:
$$Pr(\mathcal{S}) \geq \frac{2}{(ns)^2}Pr(\mathcal{M}).\square$$

Solver $\mathcal{S}$ for Case 2. In this case $\mathcal{S}$ takes input a pair $(X_0, B) \in G^2$, randomly selects one entity $\hat{B}$ from the honest entities and sets the public key of $\hat{B}$ to be B. All the other entities compute their keys normally. Furthermore, $\mathcal{S}$ randomly selects an integer $i \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows:

1. Initiate($\hat{P}_1, \hat{P}_2$): With the exception of $\hat{B}$ (whose behavior we explain below) $\hat{P}_1$ executes the Initiate() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}$ checks whether $\hat{P}_2$ is $\hat{B}$. If so, $\mathcal{S}$ sets the ephemeral public key to be X_0 from the input of $\mathcal{S}$. Otherwise, $\mathcal{S}$ aborts.
2. Respond($\hat{P}_1, \hat{P}_2, Y$): With the exception of $\hat{B}$ (whose behavior we explain below) $\hat{P}_1$ executes the Respond() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}$ checks whether $\hat{P}_2$ is $\hat{B}$. If so, $\mathcal{S}$ sets the ephemeral key to be X_0 and doesn't compute the session key. Else, $\mathcal{S}$ aborts.
3. Complete($\hat{P}_1, \hat{P}_2, X, Y$): With the exception of $\hat{B}$ (whose behavior we explain below) $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session is the i-th session at $\hat{A}$, $\mathcal{S}$ checks whether it has an open session with identifier $(\hat{A}, \hat{B}, X_0)$ and $Y = Y_0$. If so, $\mathcal{S}$ completes the session without computing a session key. Otherwise, $\mathcal{S}$ aborts.
4. $\mathcal{S}$ creates an API oracle $\mathcal{O}_{\hat{B}}$ for $\hat{B}$ as follows:

(a) When invoked to generate an ephemeral key for a session with $\hat{P}$, $\mathcal{O}_{\hat{B}}$ chooses $s, e \in Z_q$ randomly, let $Y = g^s/B^e$, define $H_1(Y, \hat{P}) = e$, and returns Y as the ephemeral public key.

(b) When invoked to compute the unhashed shared secret based on the input $(\hat{P}, X)$, $\mathcal{O}_{\hat{B}}$ returns $Z = (XP^d)^s$ where $d = H_1(X, \hat{B})$.

$\mathcal{S}$ simulates all the session activations at $\hat{B}$ for $\mathcal{M}$ with the help of $\mathcal{O}_{\hat{B}}$.

5. SessionStateReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the unhashed shared secret Z returned by the API oracle. However, if s is the i-th session at $\hat{A}$, $\mathcal{S}$ aborts.
6. SessionKeyReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$, $\mathcal{S}$ aborts.
7. Corruption($\hat{P}$): $\mathcal{S}$ gives $\mathcal{M}$ the API oracle $\mathcal{O}_{\hat{P}}$ of $\hat{P}$ and state information for current sessions and session keys at $\hat{P}$. From the moment of corruption $\mathcal{M}$ takes full control over $\hat{P}$ with the help of $\mathcal{O}_{\hat{P}}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$ when the i-th session at $\hat{A}$ is not expired, $\mathcal{S}$ aborts.
8. Expiry(s): $\mathcal{S}$ deletes the session key and any related session state of s.
9. $H_1(\cdot)$: $\mathcal{S}$ simulates a random oracle in the usual way.
10. $H(Z)$ for some Z proceeds as follows:
    - If $\mathcal{DDH}(X_0A^d, Y_0B^e) = 1$ for Z where $d = H_1(X_0, \hat{B})$ and $e = H_1(Y_0, \hat{A})$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting:

$$Z(Y_0B^e)^{-da} = g^{x_0y_0}g^{ex_0b}$$

    - $\mathcal{S}$ simulates a random oracle in the usual way.

Proof. The probability that $\mathcal{M}$ selects the i-th session of $\hat{A}$ and the peer of the test session is $\hat{B}$ is at least $\frac{1}{n^2k}$. Suppose that this is indeed the case, $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$ before $\hat{A}$'s i-th session is expired, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$, so $\mathcal{S}$ doesn't abort in Step 1, 2, 3, 5, 6, 7. So $\mathcal{S}$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability.

If $\mathcal{M}$ wins the forging attack, it computes the unhashed shared secret Z of the test session $(\hat{A}, \hat{B}, X_0, Y_0)$. Note that without the knowledge of the private key y_0 of Y_0, $\mathcal{S}$ is unable to compute CDH(X_0, B). Following the Forking Lemma [27] approach, $\mathcal{S}$ runs $\mathcal{M}$ on the same input and the same coin flips but with carefully modified answers to the H_1 queries. Note that $\mathcal{M}$ must have queried $H_1(Y_0, \hat{A})$ in its first run, because otherwise $\mathcal{M}$ would be unable to compute Z of the test session. For the second run of $\mathcal{M}$, $\mathcal{S}$ responds to $H_1(Y_0, \hat{A})$ with a value $e' \neq e$ selected uniformly at random. If $\mathcal{M}$ succeeds in the second run, $\mathcal{S}$ computes

$$Z'(Y_0B^{e'})^{-da} = g^{x_0y_0}g^{e'x_0b}$$

and thereafter obtains

$$\text{CDH}(X_0, B) = (\frac{Z}{Z'})^{\frac{1}{e-e'}}B^{-da}.$$

The forking is at the expense of introducing a wider gap in the reduction. The success probability of $\mathcal{S}$, excluding negligible terms, is

$$Pr(\mathcal{S}) \geq \frac{C}{n^2 k} Pr(\mathcal{M})$$

where C is a constant arising from the use of the Forking Lemma. □

Solver $\mathcal{S}$ for case 3. In this case $\mathcal{S}$ takes input a pair $(X_0, Y_0) \in G^2$. All the entities compute their keys normally. Furthermore, $\mathcal{S}$ randomly selects two integers $i, j \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows:

1. Initiate$(\hat{P}_1, \hat{P}_2)$: $\hat{P}_1$ executes the Initiate() activation of the protocol. If the session being created is the i-th session at $\hat{A}$, $\mathcal{S}$ checks whether $\hat{P}_2$ is $\hat{B}$. If so, $\mathcal{S}$ sets the ephemeral public key to be X_0 from the input of $\mathcal{S}$. Otherwise, $\mathcal{S}$ aborts. If the session being created is the j-th session at $\hat{B}$, $\mathcal{S}$ sets the ephemeral public key to be Y_0 from the input of $\mathcal{S}$.
2. Respond$(\hat{P}_1, \hat{P}_2, Y)$: $\hat{P}_1$ executes the Respond() activation of the protocol. If the session being created is the i-th session at $\hat{A}$, $\mathcal{S}$ checks whether $\hat{P}_2$ is $\hat{B}$ and $Y = Y_0$. If so, $\mathcal{S}$ sets the ephemeral public key to be X_0, and completes the session without computing a session key. Otherwise, $\mathcal{S}$ aborts. If the session being created is the j-th session at $\hat{B}$, $\mathcal{S}$ sets the ephemeral public key to be Y_0, then checks whether Y is generated by an oracle $\mathcal{O}_{\hat{P}}$:
   (a) If so, then $\mathcal{O}_{\hat{P}}$ must compute y and $Y = g^y$ during its run. $\mathcal{S}$ computes $s = y + dp_2$ where $d = H_1(Y, \hat{B})$, and returns $Z = (Y_0 B^e)^s$ where $e = H_1(Y_0, \hat{P}_2)$ as the return of $\mathcal{O}_{\hat{B}}$, computes the session key $K = H(Z)$, and completes the session with identifier $(\hat{B}, \hat{P}_2, Y_0, Y)$.
   (b) Else, then $\mathcal{O}_{\hat{P}}$ randomly chooses an value Z as the return of $\mathcal{O}_{\hat{B}}$, computes the session key $K = H(Z)$, and completes the session with identifier $(\hat{B}, \hat{P}_2, Y_0, Y)$.
3. Complete$(\hat{P}_1, \hat{P}_2, X, Y)$: $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}$ checks whether it has an open session with identifier $(\hat{A}, \hat{B}, X_0)$ and $Y = Y_0$. If so, $\mathcal{S}$ completes the session without computing a session key. Otherwise, $\mathcal{S}$ aborts. If the session is the j-th session at $\hat{B}$, $\mathcal{S}$ checks whether it has an open session with identifier $(\hat{B}, \hat{P}_2, Y)$ and $X = Y_0$. If fails, $\mathcal{S}$ aborts, else $\mathcal{S}$ checks whether Y is generated by some oracle $\mathcal{O}_{\hat{P}}$:
   (a) If so, then $\mathcal{O}_{\hat{P}}$ must compute y and $Y = g^y$ during its run. $\mathcal{S}$ computes $s = y + ep_2$ where $e = H_1(Y, \hat{B})$, and returns $Z = (Y_0 B^d)^s$ where $d = H_1(Y_0, \hat{P}_2)$ as the return of $\mathcal{O}_{\hat{B}}$, computes the session key $K = H(Z)$, and completes the session with identifier $(\hat{B}, \hat{P}_2, Y_0, Y)$.
   (b) Else, then $\mathcal{O}_{\hat{P}}$ randomly chooses an value Z as the return of $\mathcal{O}_{\hat{B}}$, computes the session key $K = H(Z)$, and completes the session with identifier $(\hat{B}, \hat{P}_2, Y_0, Y)$.
4. SessionStateReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the unhashed shared secrete Z. However, if s is the i-th session at $\hat{A}$, $\mathcal{S}$ aborts.

5. SessionKeyReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$, $\mathcal{S}$ aborts.
6. Corruption($\hat{P}$): $\mathcal{S}$ gives $\mathcal{M}$ the API oracle $\mathcal{O}_{\hat{P}}$ of $\hat{P}$ and state information for current sessions and session keys at $\hat{P}$. From the moment of corruption $\mathcal{M}$ takes full control over $\hat{P}$ with the help of $\mathcal{O}_{\hat{P}}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$ when the i-th session at $\hat{A}$ is not expired, $\mathcal{S}$ aborts.
7. Expiry(s): $\mathcal{S}$ deletes the session key and any related session state of s.
8. $H_1(\cdot)$: $\mathcal{S}$ simulates a random oracle in the usual way.
9. $H(Z)$ for some Z proceeds as follows:
   – If $\mathcal{DDH}(X_0A^d, Y_0B^e) = 1$ for Z where $d = H_1(X_0, \hat{B})$ and $e = H_1(Y_0, \hat{A})$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting:
   $$\mathrm{CDH}(X_0, Y_0) = ZX_0^{-eb}Y_0^{-da}g^{-deab}.$$
   – $\mathcal{S}$ simulates a random oracle in the usual way.

Proof. The probability that $\mathcal{M}$ selects the i-th session of $\hat{A}$ and the peer of the test session is $\hat{B}$ and Y_0 is generated at the j-th session at $\hat{B}$ is at least $\frac{1}{n^2k^2}$. Suppose that this is indeed the case, $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$ before $\hat{A}$'s i-th session is expired, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$, so $\mathcal{S}$ doesn't abort in Step 1, 2, 3, 4, 5, 6. So $\mathcal{S}$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. Therefore if $\mathcal{M}$ wins the forging attack, then the success probability of $\mathcal{S}$ is bounded by:

$$Pr(\mathcal{S}) \geq \frac{1}{n^2s^2}Pr(\mathcal{M}).$$ □

5.2 Infeasibility of Key-Replication Attacks

By using the GDH solver $\mathcal{S}$ above, we prove that the key-replication attacks are infeasible against HMQV by showing that such a successful adversary would break the GDH assumption.

Proof. Assume that $\mathcal{M}$ is successful in a key-replication attack against the test session $s = (\hat{A}, \hat{B}, X_0, Y_0)$. Namely, $\mathcal{M}$ succeeds in establishing a session $s' = (\hat{A}', \hat{B}', X', Y')$ which has the same key as the test session, and this session is different than $(\hat{A}, \hat{B}, X_0, Y_0)$ and $(\hat{B}, \hat{A}, Y_0, X_0)$. This means the unhashed shared secret of s and s' are same (except of a negligible probability of collision in H).

Consider the GDH solver $\mathcal{S}$ built above for the three cases. In all the three cases, $\mathcal{S}$ provides $\mathcal{M}$ (except the test session and its matching session) with the Z values of all exposed sessions. Therefore, if $\mathcal{M}$ is able to succeed in a key-replication attack then it can query the session s' (which $\mathcal{M}$ is allowed to expose) and obtains the Z of s' which equals Z of s. But this means that $\mathcal{M}$ is able to find the Z of s without exposing s or its matching session, namely, $\mathcal{M}$ can launch the forging attacks. But as we showed, in this case $\mathcal{S}$ succeeds in breaking the GDH assumption. □

This completes the proof of Lemma 2. Together with Lemma 1, we complete the proof of Theorem 1.

6 Conclusion and Future Work

We discuss the full PFS property for one-round implicitly authenticated key exchange protocols in this paper. Many works have showed that no one-round implicitly authenticated protocols can achieve full PFS, and neither the HMQV protocol. Although many solutions are proposed, they lose high performance both in communication and computation as they need to explicitly authenticate the exchanged messages. These solutions also have some limitations in the capabilities of the adversary.

We propose the idea of using tamper-proof hardware to improve the security of AKE protocols, and show that it is possible to achieve full PFS for the one-round implicitly authenticated key exchange protocols under the tamper-proof hardware assumption by formally analyzing the HMQV protocol in the CK model. Another advantage of our design of the tamper-proof hardware API for HMQV is that HMQV implemented by our design resists small group attacks even if entities don't perform the validation of ephemeral public keys.

It's interesting to investigate whether the tamper-proof hardware assumption can improve the security of other implicitly authenticated key exchange protocols. Moreover, we see that all exponentiation computations of HMQV must be performed in the hardware token, so an investigation of designing protocols requiring less computation in the hardware token could be done in the future. Another interesting work is to analyze the key exchange protocols (SM2 key exchange and MQV) in the TPM 2.0 by taking into account the protection provided by the TPM hardware. Zhao et al. [36] analyze the SM2 key exchange, and the security analysis of MQV considering the physical assumption of the TPM can be done in the future.

Acknowledgement. We appreciate anonymous SecureComm reviews for their helpful comments. This work was supported by the National Natural Science Foundation of China (91118006 and 61202414), and the National Basic Research Program of China (2013CB338003).

References

1. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
2. Boyd, C., Nieto, J.G.: On forward secrecy in one-round key exchange. In: Chen, L. (ed.) IMACC 2011. LNCS, vol. 7089, pp. 451–468. Springer, Heidelberg (2011)
3. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
4. Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg (2008)

5. Cremers, C., Feltz, M.: One-round strongly secure key exchange with perfect forward secrecy and deniability. Eidgenössische Technische Hochschule Zürich, Department of Computer Science (2011)
6. Cremers, C., Feltz, M.: Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 734–751. Springer, Heidelberg (2012)
7. Dagdelen, Ö., Fischlin, M.: Unconditionally-secure universally composable password-based key-exchange based on one-time memory tokens. Technical report, IACR Cryptology ePrint Archive (2012). http://eprint.iacr.org
8. Damgård, I.B., Nielsen, J.B., Wichs, D.: Isolated proofs of knowledge and isolated zero knowledge. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 509–526. Springer, Heidelberg (2008)
9. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**(6), 644–654 (1976)
10. Gennaro, R., Krawczyk, H., Rabin, T.: Okamoto-tanaka revisited: fully authenticated diffie-hellman with minimal overhead. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 309–328. Springer, Heidelberg (2010)
11. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
12. Goyal, V., Ishai, Y., Mahmoody, M., Sahai, A.: Interactive locking, zero-knowledge PCPS, and unconditional cryptography. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 173–190. Springer, Heidelberg (2010)
13. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
14. Huang, H.: An eCK-secure one round authenticated key exchange protocol with perfect forward security. J. Internet Serv. Inf. Secur. (JISIS) **1**(2/3), 32–43 (2011)
15. Ishai, Y., Mahmoody, M., Sahai, A.: On efficient zero-knowledge PCPs. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 151–168. Springer, Heidelberg (2012)
16. Jeong, I.R., Katz, J., Lee, D.-H.: One-round protocols for two-party authenticated key exchange. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 220–232. Springer, Heidelberg (2004)
17. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 115–128. Springer, Heidelberg (2007)
18. Kolesnikov, V.: Truly efficient string oblivious transfer using resettable tamper-proof tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 327–342. Springer, Heidelberg (2010)
19. Krawczyk, H.: HMQV: a high-performance secure diffie-hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
20. LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
21. Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 378–394. Springer, Heidelberg (2006)
22. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Des. Codes Crypt. **28**(2), 119–134 (2003)
23. Matsumoto, T., Takashima, Y.: On seeking smart public-key-distribution systems. IEICE TRANSACTIONS (1976–1990) **69**(2), 99–106 (1986)

24. Menezes, A.: Another look at HMQV. Math. Cryptology JMC **1**(1), 47–64 (2007)
25. Menezes, A., Qu, M., Vanstone, S.: Some new key agreement protocols providing mutual implicit authentication. In: Second Workshop on Selected Areas in Cryptography (SAC 95) (1995)
26. Moran, T., Segev, G.: David and goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 527–544. Springer, Heidelberg (2008)
27. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
28. RSA. PKCS# 11: Base functionality v2.30: Cryptoki - draft 4 (2009)
29. Skipjack and NIST. KEA algorithm specifications (1998)
30. TCG. Trusted platform module library part 3: Architecture family 2.0, level 00 revision 1.07 (2014)
31. TCG. Trusted platform module library part 3: Commands family 2.0, level 00 revision 1.07 (2014)
32. Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Des. Codes Crypt. **46**(3), 329–342 (2008)
33. Xu, J., Feng, D.: Comments on the SM2 key exchange protocol. In: Lin, D., Tsudik, G., Wang, X. (eds.) CANS 2011. LNCS, vol. 7092, pp. 160–171. Springer, Heidelberg (2011)
34. Yao, A.C.-C., Zhao, Y.: Oake: a new family of implicitly authenticated diffie-hellman protocols. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 1113–1128. ACM (2013)
35. Yoneyama, K.: One-round authenticated key exchange with strong forward secrecy in the standard model against constrained adversary. In: Hanaoka, G., Yamauchi, T. (eds.) IWSEC 2012. LNCS, vol. 7631, pp. 69–86. Springer, Heidelberg (2012)
36. Zhao, S., Xi, L., Zhang, Q., Qin, Y., Feng, D.: Security analysis of sm2 key exchange protocol in tpm2. 0. Security and Communication Networks (2014)