sHMQV: An Efficient Key Exchange Protocol for Power-limited Devices

Shijun Zhao and Qianying Zhang

Institute of Software Chinese Academy of Sciences,
ISCAS, Beijing, China.
zqyzsj@gmail.com, zsjzqy@gmail.com

Abstract. In this paper we focus on designing authenticated key exchange protocols for practical scenarios where the party consists of a powerful but untrusted host (e.g., PC, mobile phone, etc) and a power-limited but trusted device (e.g., Trusted Platform Module, Mobile Trusted Module, Smart Card, etc). HMQV and (s,r)OAKE[1] protocols are the state-of-the-art in the integrity of security and efficiency. However, we find that they are not suitable for the above scenarios as all (or part) of the online exponentiation computations must be performed in the power-limited trusted devices, which makes them inefficient for the deployment in practice.
To overcome the above inefficiency, we propose a variant of HMQV protocol, denoted sHMQV, under some new design rationales which bring the following advantages: 1) eliminating the validation of the ephemeral public keys, which costs one exponentiation; 2) the power-limited trusted device only performs one exponentiation, which can be pre-computed offline; 3) all the online exponentiation computations can be performed in the powerful host. The above advantages make sHMQV enjoy better performance than HMQV and (s,r)OAKE, especially when deployed in the scenarios considered in this paper. We finally formally prove the security of sHMQV in the CK model.

Keywords: Authenticated Key Exchange, CK model, Security Analysis, Power-limited Devices

1 Introduction

The authenticated key exchange (AKE) protocols aim to establish a shared secret session key between two parties via the public insecure communication while providing identity authentication. Based on the way they are authenticated, the AKE protocols can be categorized as the explicitly authenticated or the implicitly authenticated. The explicitly authenticated key exchange protocols use digital signatures or additional authenticating message flows to provide authentication and prevent replay attacks, and the typical protocol is SIGMA [4]. The implicitly authenticated key exchange protocols were first put forth by Matsumoto [18] which need only the basic Diffie-Hellman exchanges [7], yet they provide authentication by combining the ephemeral keys and long-term keys in the derivation of the session key. Due to the efficiency in both communication and computation, the implicitly authenticated key exchange protocols are widely studied and many protocols are proposed [20, 23, 17, 11, 13, 16, 15, 26, 8, 27, 5, 6, 28, 29].

For authenticated key exchange protocols, it's a basic requirement to achieve the basic security property defined by security models for key exchange, such as the CK [3] or the eCK model [15]. Besides, it's desirable to achieve the following security properties: (1) the key-compromise impersonation (KCI) resistance property; that is, the knowledge of a party's long-term private key doesn't allow the adversary to impersonate *other, uncorrupted, parties* to the party; and (2) the Perfect Forward Secrecy (PFS) property; that is, the expired session keys established before the compromise of the long-term key cannot be recovered.

[1] Yao et al. give two versions of the OAKE protocol family [28, 29], which use different terminologies to denote the protocol. In this paper, we adopt the terminologies of the full version [28], i.e., (s,r)OAKE.

1.1 Motivation

We first introduce some preliminaries used in the protocols which will be described below. Let G' be a finite Abelian group of order N, $G \subseteq G'$ be a subgroup of prime order q. Denote by g a generator of G, by 1_G the identity element, by $G \backslash 1_G = G - \{1_G\}$ the set of elements of G except 1_G and by $h = N/q$ the cofactor. The party having A as its public key will be denoted by $\hat{A}$.

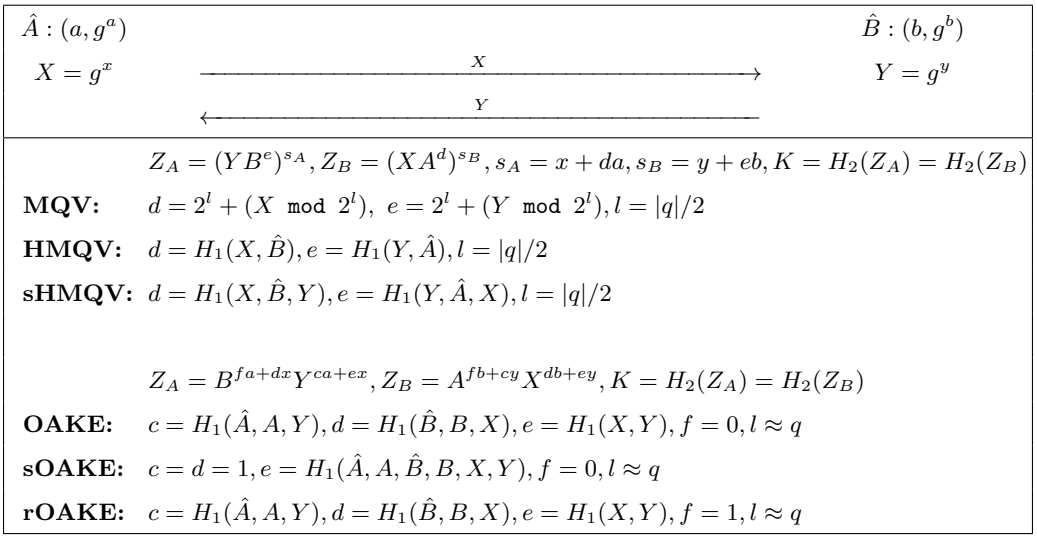

Fig. 1. (H)MQV, sHMQV, and (s,r)OAKE protocols

Up to now, many works have been done to design secure and efficient implicitly authenticated key exchange protocols. The MQV protocol was once the most efficient and most standardized [1, 2, 9, 10, 21]. Krawczyk gave a thorough analysis of MQV and found some weaknesses, then proposed a hashed variant of MQV, i.e., HMQV, which is seen as the milestone of the development of the implicitly authenticated key exchange protocols because of its efficiency and the first formal analysis of such protocols. The MQV and HMQV protocols are described in Figure 1, where H_2 (resp., H_1) is a hash function of k-bit (resp., l-bit) output and l is set to be $|q|/2$.

Yao et al. [28] consider the deployment of key exchange protocols with power-limited devices, and find that HMQV has some disadvantages in such practical contexts. First, the protocol structure of HMQV makes it not support offline pre-computing parts of the shared DH-secret (Z_A or Z_B). Second, all the online computation of the shared DH-secret must be performed in the power-limited trusted devices. As the computing power of the trusted devices might be quite limited, for example, the Trusted Platform Module (TPM) 1.2 chip only operates at 33M Hz, the disadvantages of HMQV greatly limit its deployment in such scenarios.

In order to solve the disadvantages of HMQV and provide an efficient key exchange protocol for power-limited devices, Yao et al. proposed the (s,r)OAKE protocol family [28] under new design rationales. The protocol structure of (s,r)OAKE allows the value B^{fa+dx} (resp., A^{fb+cy}) can be offline pre-computed by party $\hat{A}$ (resp., $\hat{B}$), so the online computational complexity can be only one exponentiation, i.e., Y^{ca+ex} (resp., X^{db+ey}), at each party. So it seems that (s,r)OAKE achieves the online optimal efficiency for implicitly authenticated key exchange protocols.

Just as Yao et al. said, "due to the state-of-the-art nature and highly intensive study of (H)MQV, even slight efficiency improvement can be challenging", and it becomes more challenging after (s,r)OAKE improving the online efficiency of HMQV. However, we find that (s,r)OAKE still leaves some space to improve the efficiency.

1. Both HMQV and (s,r)OAKE only consider the case where the cofactor h is small. In this case, the validation of the ephemeral public key X (resp., Y) can be reduced significantly by the embedded public-key validation technology [17], i.e., require only that $\hat{B}$ (resp., $\hat{A}$) verify that $X \in G'$ (resp., $Y \in G'$) and that $Z_B^h \neq 1_G$ (resp., $Z_A^h \neq 1_G$) before computing the session key. However, if h is large, the validation of the ephemeral public key must be performed explicitly, i.e., verify $X^q = 1_G$ that costs one exponentiation, or the protocol would be vulnerable to small subgroup attacks [19] that let an attacker recover a victim's static private key. As the validation of the ephemeral public key must be performed online, it breaks the optimal online efficiency of (s,r)OAKE.
2. The trusted device still needs to perform one expensive exponentiation online. As the computing power of the trusted device might be quite limited, such as the TPM 1.2 chip, one exponentiation computation can cause a big decrease in the total efficiency. Yao et al. proposed an efficient deployment by adopting the split mechanism proposed by Kunz-Jacques and Pointcheval [14]. Here we only show the split of the online computation of (s,r)OAKE, and please consult [28] for the split of the whole protocol. With the computation of $\hat{B}$ as an example, the trusted device computes $s_B = db + ey$ and forwards s_B to the powerful computing host, which computes $Z_B = A^{fb+cy} X^{s_B}$ and the session key. Although the split mechanism indeed improves the efficiency in practice, this deployment isn't in accordance with the formal security analysis of (s,r)OAKE: in the security analysis, the session state that is allowed to be exposed to the adversary is (y, Y, A^{cy}) but not s_B. So the security of (s,r)OAKE needs to be carefully and formally analyzed in such deployment.

The above disadvantages of HMQV and (s,r)OAKE make it inefficient for the scenario considered in this paper, i.e., the party consists of a power-limited trusted device and a powerful untrusted computing host. This leads to our motivations in the following:

1. Can we design a new protocol which doesn't need the validation of ephemeral public keys no matter how big the cofactor is, which can improve the total efficiency of the protocol by one exponentiation?
2. Can we leave all the expensive online exponentiation computations to the powerful host, which can greatly improve the online efficiency for deployment with power-limited devices in untrusted computing environment?

1.2 Our contributions

In this paper, we propose a variant of HMQV, denoted sHMQV (the s stands for schnorr), which solves the above disadvantages of HMQV and (s,r)OAKE and thus provides good performance in both total and online computational complexity. We provide detailed comparisons in Section 4.2.

1. We provide a new design rationale for key exchange protocols which eliminates the validation of the public ephemeral keys no matter how big the cofactor is: protecting the ephemeral private keys in trusted devices and designing unforgeable interfaces for trusted devices. Another advantage of our design rationale is that we formally analyze that the adversary cannot mount small subgroup attacks if public ephemeral key validation is omitted.
2. Benefited from the new design rationale, sHMQV enjoys the best total efficiency of Diffie-Hellman based key exchange protocols: 2.5 exponentiations per party no matter the cofactor is small or big. Note that if the cofactor is big, the total efficiency of HMQV is 3.5 exponentiations, and (s,r)OAKE is 4 exponentiations.
3. The trusted device performs only one exponentiation which can be offline pre-computed, and all the 1.5 online exponentiations can be performed by the powerful host. This feature makes it more desirable for being deployed with power-limited devices in untrusted powerful computing environment.

1.3 Organization

The paper is organized as follows. Section 2 outlines the CK model on which all of our analysis is based. Section 3 describes details of the interfaces of the trusted device for sHMQV, formally proves the resistance to small subgroup attacks even if the validation of the ephemeral public keys is omitted, and then describes the details of the sHMQV protocol. Section 4 introduces the design rationales of sHMQV, and compares sHMQV with related protocols in details. Section 5 formally proves sHMQV, and resistant to KCI attacks and achieves the weak PFS property in the CK model. Section 6 concludes this work and gives our future work.

2 Security Model for AKE

We outline the CK model [3] for AKE protocols, and formally describe the attacker model which models the capabilities of the adversary by some queries.

In the CK model, AKE runs in a network of interconnected parties and each party has a long-term key and a certificate (issued by a certification authority (CA)) that binds the public key to the identity of that party. A party can be activated to run an instance of the protocol called a session. Within a session a party can be activated to initiate the session or to respond to an incoming message. As a result of these activations, the party 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 party at which the session exists), a peer (the party with which the session key is intended to be established), and a session identifier. The session identifier is a quadruple $(\hat{A}, \hat{B}, out, in)$ 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 implicitly authenticated key exchange protocols, such as (s,r)OAKE and our sHMQV, 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 parties 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 parties. $\mathcal{M}$ 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. In our analysis, the session state includes the values returned by interfaces of the trusted device and the intermediate information stored and computed in the host.
- **SessionKeyReveal(s)**: $\mathcal{M}$ obtains the session key for the session s.
- **Corruption($\hat{P}$)**: This query allows $\mathcal{M}$ to learn the long-term private key of the party $\hat{P}$.
- **Test(s)**: This query may be asked only once throughout the game. Pick $b \xleftarrow{R} 0, 1$. 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". 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. A session is called *exposed* if $\mathcal{M}$ performs any one of the first 3 queries to this or the maching 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 and SessionKeyReveal 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 parties involved in the test session. Eventually $\mathcal{M}$ outputs a bit b' as its guess, then halts. $\mathcal{M}$ wins the game if $b' = b$. The adversary with the above capabilities is called a **KE-adversary**. The formal definition of security follows.

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 parties 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 Interfaces and Protocol Design

We introduce the interfaces of the trusted device which are a building block for the sHMQV protocol, and formally prove the unforgeability of the interfaces, then we formally prove that any protocols built on our interfaces can resist to small subgroup attacks. At last, we describe the details of the sHMQV protocol.

3.1 Interfaces Design and The resistance to small subgroup attacks

The trusted device stores the long-term private key of its owner, and it provides its owner two functionalities through the interfaces: 1) generating an ephemeral key, and 2) generating a Schnorr signature based on the long-term key and the ephemeral key. Figure 2 depicts the interfaces, and we give a detailed description in the following.

1. When party $\hat{A}$ wishes to establish a session key with party $\hat{B}$, its host calls the interface of its trusted device to get an ephemeral public key $X = g^x$. The ephemeral private key x is stored in the trusted device, and the public key X will be sent to $\hat{B}$.
2. After receiving the ephemeral key Y from party $\hat{B}$, $\hat{A}$ transmits $(\hat{B}, Y)$ to its trusted device through the interface, and the trusted device will perform the following steps:
   (a) Compute $d = H_1(X, \hat{B}, Y)$ and $s = x + da$ where H_1 is a hash function. d is of length $|q|/2$ where $|q|$ is the bit length of the group order.
   (b) Delete x, then return s to $\hat{A}$.

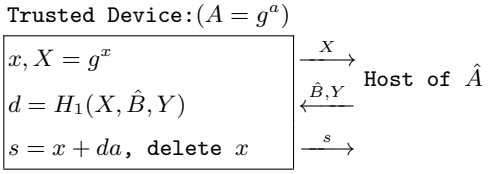

Fig. 2. The interfaces of Trusted Device

The two functionalities provided by the trusted device are actually a whole Schnorr signature (X, s) signed by the long-term key A on message $(\hat{B}, Y)$. The unforgeability of the interfaces of the trusted devices means that the adversary cannot create a legal interfaces return result which is not returned by the interfaces ever by invoking the interfaces of the trusted device at will. As it has been proven that the Schnorr signature is unforgeable against adaptive chosen message attacks in the random oracle model [22], we directly get Theorem 1. The unforgeability is based on the discrete logarithms (DL) assumption, which is explained in Appendix A.

Theorem 1. *The interfaces described above are unforgeable against adaptively chosen message attacks under the DL assumption.*

We then use the unforgeability of the interfaces to show that no adversary can successfully mount small subgroup attacks no matter whether the input ephemeral public key is validated.

Lemma 1. *If the interfaces of the trusted device are unforgeable, then the adversary cannot mount small subgroup attacks on protocols built on the interfaces.*

Proof. This lemma can be easily proved using proof by contradiction. Suppose the adversary successfully mount a small subgroup attack on some protocol, thus he obtains the long-term private key a of some party $\hat{A}$. After the adversary has obtained the long-term private key, he first computes an ephemeral key pairs $(x', X' = g^{x'})$, then computes $s' = x' + d'a$ where $d' = H_1(X', \hat{B}, Y)$. We can see that (X', s') is a legal interfaces return result on message $(\hat{B}, Y)$ forged by the adversary. This contradicts the unforgeability of the interfaces. □

Lemma 1 shows that no matter how an attacker chooses Y, for example, Y is an element of some group with small order, he cannot learn the private key a of party $\hat{A}$.

3.2 The sHMQV Protocol

Figure 3 gives an informal description of sHMQV, and the computation performed by the trusted device is boxed by rectangles.

$\hat{A} : (A = g^a)$		$\hat{B} : (B = g^b)$
$x, X = g^x$	$\xleftrightarrow{X,Y}$	$y, Y = g^y$
$d = H_1(X, \hat{B}, Y)$		$e = H_1(Y, \hat{A}, X)$
$s_A = x + da$, delete x		$s_B = y + eb$, delete y
$Z_A = (YB^e)^{s_A}$		$Z_B = (XA^d)^{s_B}$
$K_1 = H_2(Z_A, \hat{A}, \hat{B}, X, Y)$		$K_2 = H_2(Z_B, \hat{A}, \hat{B}, X, Y)$

Fig. 3. The sHMQV Protocol

We formally describe the sHMQV protocol by giving the following three session activations.

1. Initiate$(\hat{A}, \hat{B})$: $\hat{A}$ calls the interface of its trusted device 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) Verify that X is non-zero elements in the finite group (or points in a given elliptic curve).
   (b) Call the interface of its trusted device to get an ephemeral key Y, output Y as its outgoing message.
   (c) Transmit $(\hat{A}, X)$ to its trusted device and get $s_B = y + eb$ through the interface where y is the private part of Y and $e = H_1(Y, \hat{A}, X)$.
   (d) Compute $Z_B = (XA^d)^{s_B}$ where $d = H_1(X, \hat{B}, Y)$.
   (e) Compute the session key $K_2 = H_2(Z_B, \hat{A}, \hat{B}, X, Y)$ 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) Verify that Y is non-zero elements in the finite group (or points in a given elliptic curve).

(b) Transmit $(\hat{B}, Y)$ to its trusted device and get $s_A = x + da$ through the interface where x is the private part of X and $d = H_1(X, \hat{B}, Y)$.
(c) Compute $Z_A = (YB^e)^{s_A}$.
(d) Compute the session key $K_1 = H_2(Z_A, \hat{A}, \hat{B}, X, Y)$ and complete the session with identifier $(\hat{A}, \hat{B}, X, Y)$.

As d and e are of length $|q|/2$, B^e or A^d counts as "half exponentiation". Hence, the total computation of sHMQV is 2.5 exponentiations.

4 Design Rationales and Comparisons

We first describe the design rationales of sHMQV, then we make detailed comparisons between sHMQV, HMQV, (s,r)OAKE and the key exchange primitives in TPM 2.0 version [24].

4.1 Design Rationales

Protecting the ephemeral private keys. As Menezes [19] 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. So in our design of sHMQV, the ephemeral keys are generated by the trusted devices and the ephemeral private keys are protected in the trusted device. The design of (s,r)OAKE allows the exposure of ephemeral keys as they think that "the offline pre-computed and stored DH-components are less protected in practice". However, it's practical to generate and protect ephemeral keys using a hardware crypto token. Take TPM 2.0 for example, it 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() [25] command to generate an ephemeral key. Note that the above features satisfy all the properties needed by sHMQV, so protecting the ephemeral private keys by trusted devices is practical.

Design unforgeable interfaces for trusted devices. Although protecting the ephemeral keys can resist small subgroup attacks to some extent, we cannot guarantee the resistance in a formal way. Then we provide a way to formally analyze the resistance to small group attacks. First, design unforgeable interfaces for trusted devices. Then use the proof by contradiction to formally prove the resistance to the small subgroup attacks. For more details, please see section 3.1.

Omitting public key validation. Due to the above two design rationales, it's safe to omit the validation of the public keys in sHMQV, which can improve the total and online efficiency by one exponentiation.

Using Schnorr Signature. We choose the Schnorr signature as the functionality of the interfaces for two reasons: (1) it's computation ($s = x + da$) is negligible compared to the exponentiation computation, and it only costs one exponentiation if the ephemeral public key generation is considered as part of the signature, and (2) it is unforgeable and this feature is important for the security of sHMQV.

Hashing the incoming message. In HMQV, $d = H_1(X, \hat{B})$ and $e = H_1(Y, \hat{A})$. In sHMQV, the incoming message is included in the hash, i.e., $d = H_1(X, \hat{B}, Y)$ and $e = H_1(Y, \hat{A}, X)$. We show that if the incoming message is not included in the hash, sHMQV would be insecure. The adversary exposes one session, such as $(\hat{A}, \hat{B}, X, Y)$, and gets the Schnorr signature $s_A = x + da$ where $d = H_1(X, \hat{B})$. Then the adversary can always replay the message X to $\hat{B}$, and obtain the session key of $\hat{B}$ by computing $H_2(Z, \hat{A}, \hat{B}, X, Y')$ where $Z = (Y'B^e)^{s_A}$ and Y' is the ephemeral key generated by $\hat{B}$. Including the

incoming message in computing d and e can resist to replay attacks. The resistance to replay attacks makes sHMQV secure even if the adversary gets the Schnorr signature, and that's why sHMQV can be deployed in such a way that the trusted device only computes the Schnorr signature and the host computes the expensive exponentiation computations while HMQV cannot. It seems that (s,r)OAKE protocol can also be deployed in such a way. However, their security analysis isn't in accordance with such deployment. We leave the formal analysis which allows the adversary to obtain $s_A = ca + ex$ and $s_B = db + ey$ as a future work.

Session key derivation. We include the identities in the key derivation to prevent the unknown key share (UKS) attacks [12]. Moreover, we include the session identifier in the session key derivation to simplify the argument that different sessions have different session keys in our proof.

Table 1. Protocols Comparisons

		sHMQV	HMQV	(s)OAKE	rOAKE	MQV	SM2[2]
Total efficiency with (without) public key validation		2.5(2.5)	3.5(2.5)	4(3)	4(3)	3.5(2.5)	2.5(2.5)
Online efficiency with public key validation	Trusted	0	1.5	1	1	-	1.5
	Host	1.5	1	1	1	-	1
	Total	1.5	2.5	2	2	-	2.5
Online efficiency without public key validation	Trusted	0	1.5	1	1	-	1.5
	Host	1.5	0	0	0	-	0
	Total	1.5	1.5	1	1	-	1.5
Allowed secrecy exposure		(s_B, Y, Z_B)	(y, Y)	(y, Y, A^{cy})	(y, Y)	-	(y, Y, Z_B)
Assumption		GDH	GDH, KEA	GDH, KEA	GDH	-	-
Require public key validation		No	Yes	Yes	Yes	Yes	No[3]
Require POP by CA		No	No	No	No	Yes	No

4.2 Comparisons

We first make comparisons between sHMQV and the state-of-the-art protocols, i.e., HMQV, and (s,r)OAKE. We also make comparisons with the key exchange protocols implemented in TPM 2.0, i.e., the MQV and SM2 key exchange protocols [27]. The comparisons show that sHMQV is particular suitable for scenarios where a party consists of a power-limited trusted device and a powerful computing host.

Our comparisons are summarized in Table 1. The numbers denote the exponentiations. For the item of "Assumptions", the GDH and KEA stand for the Gap Decisional Diffie-Hellman and Knowledge-of-Exponent assumptions respectively.

For total efficiency, we count all the exponentiation computations per party. The numbers in parentheses is the exponentiations in the case that public key validation is not required. For online efficiency, the Trusted row counts the exponentiations performed in trusted device, the Host row counts the exponentiations performed in host, and the Total row counts the online exponentiations in total. We count online efficiency in the two cases that whether public key validation is required. The item of "Allowed secrecy exposure" refers to the information that can be exposed to the adversary. The item of "Assumption" refers to the cryptographic assumptions used in the proof. The item of "Require public key validation" refers to whether the protocol needs to validate the ephemeral public keys. The item of "Require POP by CA" refers to whether the proof of possession (POP) of private key is required by the CA. To the best of our knowledge, no work has been done for the provable security of MQV, so we use the symbol

[2] Here we refer to a patched SM2 key exchange protocol which is briefly analyzed by Xu [27].

[3] The SM2 key exchange protocol doesn't require public key validation is not because of the good design of the protocol but the restriction that it can only be used in elliptic curve groups whose cofactor is small.

"-" for the items that haven't been decided yet. Xu only gives a brief analysis of a patched SM2 key exchange protocol in [27], so the assumption of SM2 key exchange is unknown.

From Table 1, we can conclude that:

1. sHMQV enjoys the best performance no matter whether the public key validation is required or not.
2. sHMQV doesn't require trusted devices to perform any exponentiation computations, which makes it particular suitable for such applications that using TPMs to store keys. Even in general scenarios which don't consider the split of computation, sHMQV still performs very well: if public key validation is required, it enjoys the best online performance; if public validation isn't required, its online efficiency equals HMQV and is only 0.5 exponentiations more than (s,r)OAKE.
3. MQV is unpractical as the proof of possession of private key is required by the CA during public key registration.
4. Since the MQV and SM2 key exchange protocols have been adopted by the TPM 2.0 specification, which might be widely used in practice, their formal analysis work should be done as soon as possible.

5 Proof of the Protocol

In this section we analyze the security of sHMQV in the CK model. We show that it can achieve the basic security property in CK model, the weak PFS property and the resistance to KCI attacks. The security of sHMQV is based on the CDH (Computational Diffie-Hellman) and GDH (Gap Diffie-Hellman) assumptions, which are explained in Appendix A.

Session State. In order to simulate the protection of the ephemeral keys, we specify that a session state stores the results returned by the trusted devices and the information stored in the host, i.e., the Schnorr signature s and the shared secret Z before hashing.

Theorem 2. *Under the GDH assumption, the sHMQV protocol, with hash functions H_1 and H_2 modeled as random oracles, is a secure key exchange protocol and achieves the properties of weak PFS and resistance to the KCI attacks in the CK model.*

We prove theorem 2 by separately proving the basic security property, resistance to KCI attacks and the weak PFS property of sHMQV in the following three subsections respectively.

5.1 Basic Security Property

Theorem 3. *Under the GDH assumption, the sHMQV protocol, with hash functions H_1 and H_2 modeled as random oracles, is a secure key exchange protocol in the CK model.*

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

Lemma 2. *If two parties $\hat{A}$, $\hat{B}$ complete matching sessions, then their session keys are the same.*

Lemma 3. *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 2 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 $H_2(Z_A, \hat{A}, \hat{B}, X, Y)$ while $\hat{B}$ computes the same key as $H_2(Z_B, \hat{A}, \hat{B}, X, Y)$ where $Z_A = Z_B$.

The rest section proves Lemma 3. Let $\mathcal{M}$ be any adversary against the sHMQV protocol. We observe that since the session key of the test session is computed as $K = H_2(\sigma)$ for some 5-tuple σ, 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_2 on the same 5-tuple σ.
2. Key-replication attack. $\mathcal{M}$ succeeds in forcing the establishment of another session that has the same session key as the test session.

Since the session identifier $(\hat{A}, \hat{B}, X, Y)$ is hashed together with the shared secret value Z to obtain the session key K, we know that two different sessions necessarily correspond to two different session keys. What's more, since the hash function H_2 is modeled as a random oracle then the key-replication attack is impossible. So the only way for $\mathcal{M}$ to distinguish the session key K from a random value is to perform a forging attack. Next we show that if $\mathcal{M}$ can mount a successful forging attack, then we can construct a GDH solver $\mathcal{S}$ which uses $\mathcal{M}$ as a subroutine.

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 5-tuple $(Z, \hat{A}, \hat{B}, X_0, Y_0)$. 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 is 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 parties and each party participates in at most k sessions.

Solver $\mathcal{S}$ for case 1 In this case $\mathcal{S}$ takes input a pair $(X_0, Y_0) \in G^2$, creates an AKE experiment which includes n parties, and is given access to a DDH oracle $\mathcal{DDH}$. $\mathcal{S}$ assigns the n parties random static key pairs, then randomly selects two integers $i, j \in [1, ..., k]$ and two honest parties $\hat{A}$ and $\hat{B}$. $\mathcal{S}$ runs sHMQV 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 i-th session at $\hat{A}$ (or 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 i-th session at $\hat{A}$ (or 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 i-th session at $\hat{A}$ (or j-th session at $\hat{B}$), $\mathcal{S}$ checks whether $Y = Y_0$ (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 Schnorr signature σ returned by the oracle and the shared secrete Z. However, if s is i-th session at $\hat{A}$ (or 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 i-th session at $\hat{A}$ (or j-th session at $\hat{B}$), $\mathcal{S}$ aborts.
6. Corruption$(\hat{P})$: $\mathcal{S}$ gives $\mathcal{M}$ the private key 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 $\hat{P}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}$ aborts.
7. $H_1(\cdot)$: $\mathcal{S}$ simulates a random oracle in the usual way.
8. $H_2(\sigma)$ for some $\sigma = (Z, \hat{P}_1, \hat{P}_2, X, Y)$ proceeds as follows:

- If $\sigma = (Z, \hat{A}, \hat{B}, X_0, Y_0)$ for some Z value, and $\mathcal{DDH}(X_0A^d, Y_0B^e) = 1$ where $d = H_1(X_0, \hat{B}, Y_0)$ and $e = H_1(Y_0, \hat{A}, X_0)$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting:
$$\text{CDH}(X_0, Y_0) = ZX_0^{-eb}Y_0^{-da}g^{-deab}.$$
- If the value of the function on input σ has been previously defined, return it.
- If not defined, $\mathcal{S}$ simulates a random oracle as usual.

The probability that $\mathcal{M}$ selects i-th session of $\hat{A}$ and 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 $\hat{A}$ and $\hat{B}$, 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:

$$Pr(\mathcal{S}) \geq \frac{2}{(ns)^2} Pr(\mathcal{M}).$$

Solver $\mathcal{S}$ for case 2 In this case $\mathcal{S}$ takes input a pair $(X_0, B) \in G^2$, randomly selects one party $\hat{B}$ from the honest parties and sets the public key of $\hat{B}$ to be B. All the other parties 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 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 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 i-th session at $\hat{A}$, $\mathcal{S}$ completes the session without computing a session key.
4. $\mathcal{S}$ creates an oracle $\mathcal{O}_{\hat{\mathcal{B}}}$ as follows:
   (a) When invoked to generate an ephemeral key, $\mathcal{O}_{\hat{\mathcal{B}}}$ chooses $s, e \in Z_q$ randomly, let $Y = g^s/B^e$, and returns Y as the ephemeral public key.
   (b) When invoked to generate a Schnorr signature on an input message $(\hat{P}, X)$, $\mathcal{O}_{\hat{\mathcal{B}}}$ sets $H_1(Y, \hat{P}, X) = e$, and returns (Y, s) as the Schnorr signature. If $H_1(Y, \hat{P}, X)$ was defined as $e' \neq e$ by a previous query to H_1, $\mathcal{S}$ rewinds the computation up to the point in which $H_1(Y, \hat{P}, X)$ was defined as e', and this time define $H_1(Y, \hat{P}, X) = e$.

   $\mathcal{S}$ simulates all the session activations at $\hat{B}$ for $\mathcal{M}$ with the help of $\mathcal{O}_{\hat{\mathcal{B}}}$.
5. SessionStateReveal(s): $\mathcal{S}$ returns to $\mathcal{M}$ the Schnorr signature σ returned by the oracle and the shared secrete Z. However, if s is 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 i-th session at $\hat{A}$, $\mathcal{S}$ aborts.
7. Corruption$(\hat{P})$: $\mathcal{S}$ gives $\mathcal{M}$ the private key 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 $\hat{P}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}$ aborts.
8. $H_1(\cdot)$: $\mathcal{S}$ simulates a random oracle in the usual way.
9. $H_2(\sigma)$ function for some $\sigma = (Z, \hat{P}_1, \hat{P}_2, X, Y)$ proceeds as follows:
   - If $\sigma = (Z, \hat{A}, \hat{B}, X_0, Y_0)$ for some Z value, and $\mathcal{DDH}(X_0A^d, Y_0B^e) = 1$ where $d = H_1(X_0, \hat{B}, Y_0)$ and $e = H_1(Y_0, \hat{A}, X_0)$, then $\mathcal{S}$ aborts $\mathcal{M}$ and is successful by outputting:
   $$Z(Y_0B^e)^{-da} = g^{x_0y_0}g^{ex_0b}$$
   - If the value of the function on input σ has been previously defined, return it.

– If not defined, $\mathcal{S}$ simulates a random oracle as usual.

The probability that $\mathcal{M}$ selects 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 $\hat{B}$, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$, so $\mathcal{S}$ doesn't abort in Step 1, 2, 5, 6, 7. So $\mathcal{S}$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability.

If $\mathcal{M}$ wins the forging attack, he computes the 5-tuple $(Z, \hat{A}, \hat{B}, X_0, Y_0)$. 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 [22] 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 $H_1(Y_0, \hat{A}, X_0)$ 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 $H_1(Y_0, \hat{A}, X_0)$ with a value $e' \neq e$ selected uniformly at random. If $\mathcal{M}$ succeeds in the second run, $\mathcal{S}$ computes

$$Z'(Y_0 B^{e'})^{-da} = g^{x_0 y_0} g^{e' x_0 b}$$

and thereafter obtains

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

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

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

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

Solver $\mathcal{S}$ for case 3 The simulation for $\mathcal{M}$'s environment for case 3 is the same as case 2. Now we show that if $\mathcal{M}$ mounts a successful forging attack on sHMQV in this case, $\mathcal{S}$ could break the GDH assumption.

By the same argument in case 2, $\mathcal{S}$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. If $\mathcal{M}$ wins the forging attack, he computes the 5-tuple $(Z, \hat{A}, \hat{B}, X_0, Y_0)$, where

$$Z = g^{(x_0 + da)(y_0 + eb)} \tag{1}$$

and $d = H_1(X_0, \hat{A}, Y_0)$ and $e = H_1(Y_0, \hat{B}, X_0)$. Since Y_0 was generated at $\hat{B}$ during a session $(\hat{B}, \hat{A}^*, Y_0, X^*)$ with $\hat{A}^* \neq \hat{A}$ or $X^* \neq X_0$, the oracle $\mathcal{O}_{\hat{B}}$ simulated by $\mathcal{S}$ must output a Schnorr signature (Y_0, s) with $Y_0 = g^s / B^{e'}$ where $e' = H_1(Y, \hat{A}^*, X^*)$. Since $(\hat{A}^*, X^*) \neq (\hat{A}, X_0)$, then $e' \neq e$. Let

$$Z' = (X_0 A^d)^s = g^{(x_0 + da)(y_0 + e'b)} \tag{2}$$

Dividing equation 1 by equation 2 we obtain that

$$\text{CDH}(X_0, B) = g^{x_0 b} = (\tfrac{Z}{Z'})^{\frac{1}{e-e'}} B^{-da}.$$

Therefore if $\mathcal{M}$ wins the forging attack, then the success probability of $\mathcal{S}$ is bounded by:

$$Pr(\mathcal{S}) \geq \tfrac{1}{n^2 k} Pr(\mathcal{M}).$$

This completes the proof of Lemma 3. Together with Lemma 2, we complete the proof of Theorem 3.

5.2 Resistance to KCI attacks

Consider the practical scenarios in which the private key of $\hat{A}$ is compromised, yet the adversary doesn't actively control $\hat{A}$ (e.g., it does not have access to $\hat{A}$'s computer). The KCI resistance ensures that in such scenarios the session keys of $\hat{A}$ cannot be compromised by $\mathcal{M}$. In particular, $\mathcal{M}$ cannot impersonate an uncorrupted party $\hat{B}$ to $\hat{A}$.

Theorem 4. *Under the GDH assumption, the sHMQV protocol, with hash functions H_1 and H_2 modeled as random oracles, resists key-compromise impersonation attacks.*

Proof. All is needed is to note that (1) since in the proof of Theorem 3 the outgoing value X_0 of $\hat{A}$ in the test session is provided by the input to $\mathcal{S}$ and then not chosen by $\mathcal{M}$, then $\mathcal{M}$ doesn't actively control $\hat{A}$, and (2) the proof of Theorem 3 holds even if we assume that the adversary $\mathcal{M}$ is given the private key of $\hat{A}$. We change the GDH solver $\mathcal{S}$ described in the proof of Theorem 3 by removing the corruption of $\hat{A}$ as a reason for $\mathcal{S}$ to abort in all the 3 cases described in Section 5.1. The proof remains valid since the simulation is still perfect and the above abort operations are never used in the proof. □

5.3 Weak Perfect Forward Secrecy

The weak PFS ensures that the session key of a session whose incoming and outgoing messages X, Y are not chosen by the adversary enjoys forward secrecy.

Theorem 5. *Under the CDH assumption, the sHMQV protocol, with hash functions H_1 and H_2 modeled as random oracles, provides weak PFS property.*

Proof. We only sketch the proof as the proof can be easily completed following the proof in Section 5.1. Given an adversary $\mathcal{M}$ that breaks the weak PFS property, we construct a CDH solver $\mathcal{C}$ as follows. Let $(X_0 = g^{x_0}, Y = g^{y_0})$ be the input pair to $\mathcal{C}$, and the goal of $\mathcal{C}$ is to compute $g^{x_0 y_0}$. $\mathcal{C}$ runs $\mathcal{M}$ in a simulated environment in which $\mathcal{C}$ chooses all parties' private keys. We assume that the environment involves at most n parties and each party participates in at most k sessions. $\mathcal{C}$ chooses $i \in [1...k]$, and two parties $\hat{A}$ and $\hat{B}$ whose private keys are a and b respectively. $\mathcal{C}$ uses X_0, Y_0 as the incoming and outgoing messages in the i-th session of $\hat{A}$. If $\mathcal{M}$ selects i-th session of $\hat{A}$ as the test session and its peer is $\hat{B}$ (with the probability at least $\frac{1}{n^2 k}$) and distinguishes the session key of the test session from random, then $\mathcal{M}$ must query H_2 with the 5-tuple $(Z, \hat{A}, \hat{B}, X_0, Y_0)$ where $Z = g^{(x_0+da)(y_0+eb)}$ at some point, i.e., $\mathcal{M}$ must output Z in its run. With the knowledge of a and b, $\mathcal{C}$ can compute $g^{x_0 y_0}$. □

6 Conclusions and Future Work

In this paper, we propose new design rationales for implicitly authenticated key exchange protocols, which can eliminate the public key validation and design protocols particular suitable for scenarios where the party consists of a power-limited trusted device and a powerful but untrusted computing host. Based on the new design rationales, we propose the sHMQV protocol, which performs the best total efficiency compared to the state-of-the-art protocols and leaves all the online exponentiation computations to the host.

As we have mentioned above, it seems that our design rationales can apply to the (s,r)OAKE protocol, so it's interesting to formally analyze (s,r)OAKE under our design rationales. Another work that should to be done is the security analysis of the MQV and SM2 key exchange protocols, which have been adopted by the TPM 2.0 specification.

References

1. American National Standard (ANSI) X9.42-2001. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.
2. American National Standard (ANSI) X9.63. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography.
3. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, pages 453–474. Springer, 2001.
4. R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *Advances in Cryptology – CRYPTO 2002*, pages 143–161. Springer, 2002.
5. C. Cremers and M. Feltz. *One-Round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science, 2011.
6. C. Cremers and M. Feltz. Beyond eCK: Perfect Forward Secrecy under Actor Compromise and Ephemeral-Key Reveal. In *Computer Security – ESORICS 2012*, pages 734–751. Springer, 2012.
7. W. Diffie and M. Hellman. New Directions in Cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.
8. R. Gennaro, H. Krawczyk, and T. Rabin. Okamoto-Tanaka revisited: Fully authenticated Diffie-Hellman with minimal overhead. In *Applied Cryptography and Network Security*, pages 309–328. Springer, 2010.
9. IEEE 1363-2000. Standard Specifications for Public Key Cryptography.
10. ISO/IEC IS 15946-3. Information Technology - Security Techniques - Cryptographic Techniques Based on Elliptic Curves - Part 3: Key Establishment, 2002.
11. I. R. Jeong, J. Katz, and D. H. Lee. One-Round Protocols for Two-Party Authenticated Key Exchange. In *Applied Cryptography and Network Security*, pages 220–232. Springer, 2004.
12. B. S. Kaliski Jr. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):275–288, 2001.
13. H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.
14. S. Kunz-Jacques and D. Pointcheval. A New Key Exchange Protocol Based on MQV Assuming Public Computations. In *Security and Cryptography for Networks*, pages 186–200. Springer, 2006.
15. B. LaMacchia, K. Lauter, and A. Mityagin. Stronger Security of Authenticated Key Exchange. In *Provable Security*, pages 1–16. Springer, 2007.
16. K. Lauter and A. Mityagin. Security Analysis of KEA Authenticated Key Exchange Protocol. In *Public Key Cryptography – PKC 2006*, pages 378–394. Springer, 2006.
17. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.
18. T. Matsumoto and Y. Takashima. On Seeking Smart Public-Key-Distribution Systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.
19. A. Menezes. Another look at HMQV. *Mathematical Cryptology JMC*, 1(1):47–64, 2007.
20. A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentication. In *Second Workshop on Selected Areas in Cryptography (SAC 95)*, 1995.
21. NIST Special Publication 800-56 (DRAFT). Recommendation on Key Establishment Schemes, January 2003.
22. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – EUROCRYPT'96*, pages 387–398. Springer, 1996.
23. Skipjack and NIST. KEA algorithm specifications, 1998.
24. TCG. Trusted Platform Module Library Part 1: Architecture, Family 2.0, Level 00 Revision 01.07, 2014.
25. TCG. Trusted Platform Module Library Part 3: Commands Family 2.0, Level 00 Revision 01.07, 2014.
26. B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Designs, Codes and Cryptography*, 46(3):329–342, 2008.
27. J. Xu and D. Feng. Comments on the SM2 key exchange protocol. In *Cryptology and Network Security*, pages 160–171. Springer, 2011.
28. A. C. Yao and Y. Zhao. A New Family of Implicitly Authenticated Diffie-Dellman Protocols. Technical report, Cryptology ePrint Archive, Report 2011/035, 201 1. http://eprint. iacr. org/.(Cited on pages 10 and 15.).
29. A. C.-C. Yao and Y. Zhao. OAKE: A New Family of Implicitly Authenticated Diffie-Dellman Protocols. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1113–1128. ACM, 2013.

A Preliminaries

In this section we define the assumptions that are used in this paper.

Definition 2. *Let G be a cyclic group of order p with generator g. The DL assumption in G states that, given $A = g^a \in G$, it is computationally infeasible to compute the discrete logarithm a of A.*

Definition 3. *Let G be a cyclic group of order p with generator g. The CDH assumption in G states that, given two randomly chosen points $X = g^x$ and $Y = g^y$, it is computationally infeasible to compute $Z = g^{xy}$.*

Definition 4. *Let G be a cyclic group generated by an element g whose order is p. We say that a decision algorithm $\mathcal{O}$ 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 $\mathcal{O}$ 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.*