A Unified Security Analysis of Two-phase Key Exchange Protocols in TPM 2.0

(Full Version)[⋆]

Shijun Zhao[1] and Qianying Zhang[2]

[1] TCA lab, Institute of Software Chinese Academy of Sciences,
Beijing 100190, China.
zqyzsj@gmail.com

[2] College of Information Engineering, Capital Normal University,
Beijing 100048, China
zsjzqy@gmail.com

Abstract. The Trusted Platform Module (TPM) version 2.0 provides an authenticated key exchange functionality by a single key exchange primitive, which can be called to implement three key exchange protocols (denoted as two-phase key exchange protocols in TPM 2.0): the Full Unified Model, the MQV, and the SM2 key exchange protocols. However, some vulnerabilities have been found in all of these protocols. Fortunately, it seems that protections provided by the TPM can deal with vulnerabilities of these protocols. This paper investigates whether the TPM key exchange primitive provides a secure key exchange functionality under protections of the TPM. We first perform an informal analysis of the TPM key exchange primitive which helps us to model in a precise way. Then we formally analyze the TPM key exchange primitive in a security model for AKE, based on which all the protocols adopted by TPM 2.0 can be analyzed in a unified way. Our analysis indicates under what conditions the TPM 2.0 can provide a provable secure key exchange functionality. In the end, we give suggestions on how to leverage the TPM key exchange primitive properly, and suggestions on how to improve the security of current TPM key exchange primitive to enable its wide use in practice.

Keywords: Authenticated Key Exchange, Security Model, Security Analysis, Min-entropy, TPM 2.0

1 Introduction

Authenticated key exchange (AKE) is a very important public key primitive in modern cryptography, which allows two parties to establish a shared secret session key via the public insecure communication while providing mutual authentication. To achieve security against active attackers, who control the public communication channels, AKE protocols commonly use digital signatures or message authentication codes (MAC) to explicitly authenticate the messages exchanged. Some typical examples include: STS [5], SIGMA [8], TLS [11], and JFK [3]. However, the authentication mechanism to resist active attacks incurs a significant increase in both the computation and communication complexity compared to the basic Diffie-Hellman key exchange protocol.

In 1986, Matsumoto et al. first put forth the design of implicitly AKE protocols [21] which only required basic Diffie-Hellman exchanges while providing identities authentication by combining the ephemeral keys and long-term keys in the derivation of the session key. The implicitly AKE protocols

[⋆] An extended abstract of this paper appears in Trust'15.

$$\begin{array}{lcr} \hat{A} : (a, A = g^a) & & \hat{B} : (b, B = g^b) \\ X = g^x & \xrightarrow{\quad\quad\quad X \quad\quad\quad} & Y = g^y \\ & \xleftarrow{\quad\quad\quad Y \quad\quad\quad} & \end{array}$$

$$Z_A = (YB^e)^{h(x+da)}, Z_B = (XA^d)^{h(y+eb)}$$

Full Unified Model: $K = H_1(Z_1, Z_2, \hat{A}, \hat{B}, X, Y)$, where $Z_1 = g^{ab}$, $Z_2 = g^{xy}$

Full MQV: $K = H_2(Z_A, \hat{A}, \hat{B}) = H_2(Z_B, \hat{A}, \hat{B})$, where

$$d = avf(X) \overset{\text{def}}{=} 2^l + (X.x \bmod 2^l), e = avf(Y) \overset{\text{def}}{=} 2^l + (Y.x \bmod 2^l), l = \lceil q/2 \rceil$$

SM2 Key Exchange: $Z_A = (BY^e)^{h(a+dx)}, Z_B = (AX^d)^{h(b+ey)}$

$$K = H_2(Z_A, \hat{A}, \hat{B}) = H_2(Z_B, \hat{A}, \hat{B}), \text{ where}$$

$$d = avf'(X) \overset{\text{def}}{=} 2^l + (X.x \bmod 2^l), e = avf'(Y) \overset{\text{def}}{=} 2^l + (Y.x \bmod 2^l), l = \lfloor q/2 \rfloor$$

Fig. 1. The Full Unified Model, Full MQV, and SM2 key exchange Protocols

achieve efficiency in both communication and computation, so they are widely studied and many protocols are proposed [22, 25, 20, 14, 17, 19, 18, 29, 13, 31–33]. Among these protocols, the HMQV protocol [17] marks the milestone of the development of the implicitly AKE protocols because it provides the first formal analysis of implicitly AKE protocols within a modern AKE security model (the CK model). By now, it becomes a basic requirement for AKE protocols to achieve the security defined by modern AKE security models, such as the CK [7] model or eCK [18] model. The core security property defined by modern AKE security models guarantees that the corruption of one session would not compromise the security of other sessions. In modern AKE security models, a session of implicitly AKE protocols is identified by a quadruple $(\hat{A}, \hat{B}, X, Y)$ where $\hat{A}$ is the identity of the holder of the session, $\hat{B}$ the peer, X the outgoing ephemeral public key, and Y the incoming ephemeral public key.

In this work, we focus on the two-phase key exchange primitive defined in the new released TPM 2.0 specification [27], which supports three implicitly AKE protocols: the Full Unified Model and Full MQV protocols described in SP800-56A [4], and the SM2 key exchange protocol [31]. The three protocols are described as two-phase key exchange protocols in TPM 2.0 as they require two phases. In the first phase, the TPM generates an ephemeral DH key to be sent to the other party. In the second phase, the TPM generates the unhashed shared secret by combining ephemeral keys and long-term keys, and then the host of the TPM uses the unhashed shared secret to derive the session key.

We first introduce some preliminaries used in the three protocols. 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. We use multiplicative notation for the group operation in G'. Let $u \in_R Z_q$ denote randomly selecting an integer u between 1 and $q - 1$. Note that G actually is an elliptic curve in this work as all the three protocols are based on elliptic curve cryptography. Let $P.x$ denote the x-coordinate of point P. The party having A as its public key will be denoted by $\hat{A}$. The Full Unified Model, Full MQV and SM2 key exchange protocols are described in Figure 1. $H_1()$ and $H_2()$ are cryptographic hash functions. The Full Unified Model protocol analyzed in this paper includes the ephemeral public keys exchanged as suggested by [15]. The Full MQV protocol is a variant of the original MQV protocol [22] (which doesn't include parties' identifiers in the session key derivation, i.e., $K = H_2(Z_A) = H_2(Z_B)$).

1.1 Weaknesses of AKE protocols in TPM 2.0

Unfortunately, all the three AKE protocols adopted by TPM 2.0 are not secure[1]. We summarize their weaknesses in the following.

We find that the Full Unified Model key exchange protocol is completely insecure if an attacker is able to learn the intermediate information $Z_1 = g^{ab}$ of some session established by $\hat{A}$ with $\hat{B}$: the attacker transmits an ephemeral key $X' = g^{x'}$ generated by himself to party $\hat{B}$ and receives an ephemeral public key Y' from $\hat{B}$, then he can compute the session key $K = H(Z_1, Y'^{x'}, \hat{A}, \hat{B}, X', Y')$, i.e., the attacker is able to impersonate $\hat{A}$ to $\hat{B}$ indefinitely.

Kaliski presented an unknown-key share (UKS) attack [16] on the original MQV protocol in which the attacker $\mathcal{M}$ interfaces with the session establishment between two honest parties $\hat{A}$ and $\hat{B}$ such that $\hat{A}$ is convinced that he is sharing a key with $\hat{B}$, but $\hat{B}$ believes that he is sharing the same session key with $\mathcal{M}$. $\mathcal{M}$ can mount Kaliski's UKS attack by (a) registering with the CA a specific key $C = g^c$, and (b) sending a specific ephemeral public key X' to $\hat{B}$. c and X' are cleverly computed by $\mathcal{M}$ such that session keys of sessions $(\hat{A}, \hat{B}, X, Y)$ and $(\hat{B}, \mathcal{M}, Y, X')$ are identical. A detailed description of the UKS attack can be found in Appendix A. Although the Full MQV protocol tries to overcome the UKS weakness by including identities in the session key derivation, we find that it still cannot achieve the security defined by modern AKE models if $\mathcal{M}$ is able to learn the unhashed shared Z value: $\mathcal{M}$ performs the same steps above, learns Z_B by corrupting the session $(\hat{B}, \mathcal{M}, Y, X')$, then $\mathcal{M}$ can compute the session key of session $(\hat{A}, \hat{B}, X, Y)$, i.e., corruption of the session $(\hat{B}, \mathcal{M}, Y, X')$ helps $\mathcal{M}$ to compromise another session $(\hat{A}, \hat{B}, X, Y)$. In the following of this paper, we use MQV to denote the Full MQV.

Xu et al. introduced two attacks [31] on the SM2 key exchange protocol in which an honest party $\hat{A}$ is coerced to share a session key with the attacker $\mathcal{M}$, but $\hat{A}$ thinks that he is sharing the key with another party $\hat{B}$. Both attacks requires $\mathcal{M}$ to reveal the unhashed shared Z_B in $\hat{B}$. Besides, the first attack requires $\mathcal{M}$ to register with the CA a specific key $C = Ag^e$ where $e \in_R Z_q$, and the second attack requires $\mathcal{M}$ to perform some computations using his private key after obtaining Z_B. A detailed description of the two attacks can be found in Appendix B.

From above attacks we can see that the three AKE protocols cannot achieve the security property defined by modern AKE security models if the attacker is able to get the unhashed Z values. Unfortunately, this is exactly how the TPM 2.0 two-phase key exchange primitive implements these three AKE protocols: Z_1 of the Full Unified Model, unhashed Z value of the MQV and SM2 key exchange protocols are returned to the host, whose memory is vulnerable to attacks. So it seems that the TPM 2.0 key exchange primitive is not secure.

1.2 Motivations and Contributions

Fortunately, protections provided by the TPM improve the security of the TPM key exchange primitive. We use tpm.KE to denote the TPM key exchange primitive in this paper. First, all long-term keys are generated by TPM chip randomly, so the attacker cannot use the TPM chip to generate a specific key such as the cleverly computed key $C = g^c$ in Kaliski's UKS attack or $C = Ag^e$ in Xu's first attack. Second, the TPM only provides fixed functionalities through TPM commands [28] in a black-box manner: when a TPM command is invoked, the TPM chip executes the pre-defined computation procedure, and returns the computation result. The second feature constrains the attacker from using the key to perform computations at will. It seems that above two features can prevent Kaliski's UKS attack and Xu's attacks, and Zhao et al. [34] show that protections provided by the TPM indeed help the SM2 key exchange protocol to resist Xu's two attacks by an informal analysis. However, Zhao et al. don't model the two features above in their formal analysis, but perform their formal analysis by adopting an

[1] The TPM 2.0 specification notes that the Full MQV and SM2 key exchange protocols "may be susceptible to unknown key-share (UKS) attacks" [27].

easier approach: they modify tpm.KE not to return the unhashed Z value but the session key, thus Z is not available to the attacker. This leads to our first motivation:

1. *How to precisely model protections provided by the TPM, and check whether the protections can help current* tpm.KE *to be proven secure?*

Although protections provided by the TPM help the MQV and SM2 key exchange protocols to resist current attacks, the $avf()$ and $avf'()$ used in the MQV and SM2 key exchange protocols respectively make that no analysis can prove these two protocols to be secure. Consider such a group G that the representation of its elements satisfies that the $\lceil q/2 \rceil$ least significant bits (LSBs) of the representation of points' x-coordinate are fixed. In this case, the attacker can mount the so-called group representation attacks on MQV and SM2 key exchange protocols, in which the attacker can impersonate $\hat{A}$ to $\hat{B}$ without knowing the private key of $\hat{A}$. A group representation attack on MQV is described in Appendix C, and a similar attack on the SM2 key exchange protocol can be found in [34]. To make this type of attacks more convincing, [34] proposes an approach to construct such an elliptic curve in theory. HMQV, a variant of MQV proved secure in the CK model, prevents this type of attack by replacing $avf()$ with a cryptographic hash function. [34] also suggests replacing the $avf'()$ of the SM2 key exchange protocol with a cryptographic hash function. However, group representation attacks are not practical as in practice it's difficult to find an elliptic curve whose $\lceil q/2 \rceil$ LSBs of the representation of points' x-coordinate are fixed. On the contrary, the generation of the $avf()$ and $avf'()$ seems to range in a uniform way over all possible values. This leads to our second motivation:

2. *Can we give a quantitative measure of the amount of randomness (entropy) contained in the practical output distribution of $avf()$ and $avf'()$, and check whether $avf()$ and $avf'()$ provide enough entropy to prevent group representation attacks?*

The tpm.KE is designed to support three implicitly AKE protocols through a unified interface. However, current modern AKE security models only consider how to formally analyze one single protocol. To the best of our knowledge, all AKE protocols proven secure in the literature are analyzed separately. For example, [34] only analyzes the SM2 key exchange protocol in TPM 2.0, and doesn't model the other two protocols. However, this analysis approach is insufficient for tpm.KE. Suppose an honest party $\hat{A}$ tries to establish a secure channel with $\hat{B}$ through MQV, and the TPM of $\hat{B}$ has a long-term key of the type SM2, which is controlled by the attacker. Is the session key of $\hat{A}$ still secure if the attacker leverages the key of the type SM2 to complete the session? In this case, it's desirable for tpm.KE to protect the session key of $\hat{A}$. We denote this security property by *correspondence property*. However, current security models don't capture this security property. This leads to our third motivation:

3. *Can we build a unified security AKE model, based on which we can give a formal analysis of* tpm.KE*, which supports three AKE protocols?*

Contributions. We summarize the contributions of this paper as follows:

1. We leverage the min-entropy, a notion from information theory, to give a quantitative measure of the amount of randomness in the output distribution of $avf()$ and $avf'()$. We measure several series of elliptic curves used in practice, covering all elliptic curves adopted by TPM 2.0 [26]. We also compare the measurement with a cryptographic hash function, SHA-2. The comparison results show that $avf()$ and $avf'()$ provide almost the same level of randomness as cryptographic hash functions.
2. We model the protections provided by the TPM by modeling the interfaces of tpm.KE as oracles, and present a unified AKE security model for tpm.KE, which captures not only the security property defined by modern AKE models but also the correspondence property.
3. We give a formal analysis of tpm.KE in our new model, and prove that tpm.KE is secure under the condition that the unhashed shared secrets are not available to the attacker. This condition can be achieved by slightly modifying the Full Unified Model functionality of TPM 2.0 or proper implementation of the host's software which derives the session key.

4. The tpm.KE defined by current TPM 2.0 specification opens a window of opportunity to actually mount impersonate attacks, so we give suggestions on how to avoid such attacks. We also give some suggestions on how to modify TPM 2.0 specification to achieve a more secure tpm.KE.

1.3 Organization

In the rest of this paper, Section 2 gives some preliminaries. Section 3 introduces the two-phase key exchange primitive defined by TPM 2.0 specification, gives a quantitative measure of several series of elliptic curves used in practice, and presents an informal analysis of tpm.KE. Section 4 presents our unified security model for tpm.KE. Section 5 gives a formal description of tpm.KE. Section 6 proves the unforgeabilities of the functionalities of MQV and SM2 key exchange provided by tpm.KE, which can simplify our analysis. Section 7 formally analyzes tpm.KE in our new model. Section 8 discusses some further security properties, and gives our suggestions on how to implement secure AKE protocols based on current tpm.KE and how to modify current TPM 2.0 specification to achieve a more secure key exchange primitive. Section 9 concludes this paper and gives our future work.

2 Preliminaries

This section first introduces the notion of min-entropy and two commonly used methods to calculate the min-entropy, then introduces CDH (Computational Diffie-Hellman) and GDH (Gap Diffie-Hellman) assumptions used in this paper.

2.1 Min-entropy

Min-entropy is a notion from information theory, which provides a very strict information-theoretical lower bound (i.e., worst-case) measure of randomness for a random variable. High min-entropy indicates that the distribution of the random variable is close to the uniform distribution. Low min-entropy indicates that there must be a small set of outcomes that has an unusually high probability, and the small set can help the attacker to perform the group representation attack. Take the two extreme cases for example: if the min-entropy of a random variable is equal with the length of the outcome, the distribution is a uniform distribution, and if the min-entropy of a random variable is zero, the outcomes of the random variable are a fixed value. From the two extreme cases we can see that the higher the min-entropy is, the harder for the attacker to mount group representation attacks. There are usually two methods to measure the min-entropy of a random variable:

1. NIST SP 800-90. This method is described in NIST specification 800-90 for binary sources. The definition for min-entropy of one binary bit is: $H = -log_2(p_{max})$, where $p_{max} = max\{p_0, p_1\}$, and p_0, p_1 are probabilities of the binary bit outputs zero and one respectively. The min-entropy of an n-bit binary string is defined by:
$$H_{total} = \sum_{i=1}^{n} H_i \tag{1}$$
2. Context-Tree Weighting compression. Context-Tree Weighting (CTW) [30] is an optimal compression algorithm for stationary sources and is commonly used for estimate the min-entropy.

2.2 CDH and GDH Assumptions

Definition 1 (CDH Assumption). *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 2 (GDH Assumption). *Let G be a cyclic group generated by an element g whose order is p. We say that a decision algorithm $\mathcal{O}$ 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.*

3 The TPM Key Exchange Primitive

This section first presents how tpm.KE is implemented in TPM 2.0 and introduces relevant TPM commands. Then we give an informal analysis of tpm.KE. In our informal analysis, we present our solutions to prevent impersonation attacks on the Full Unified Model protocol, and a quantitative measure of the randomness of the output distribution of $avf()$ and $avf'()$ on a wide range of elliptic curves which have been widely used in practice.

3.1 Introduction of tpm.KE

tpm.KE consists of two phases. In the first phase, the TPM generates an ephemeral key which is transferred to the other party. In the second phase, the TPM generates the unhashed secret values according to the specification of the selected protocol, then the host derives the session key from the unhashed secret values. Before running the two phases, the **Key Generation** procedure should be invoked first to generate the long-term key. As we aim to analyze the whole AKE protocols adopted by TPM 2.0, tpm.KE introduced below not only includes the key exchange functionality provided by the TPM, but also the session key derivation procedure performed on the host.

Key Generation The relevant commands are TPM2_Create() and TPM2_CreatePrimary(). They take as input public parameters including an attribute identifying the key exchange scheme for the long-term key. The scheme should be one of the following three: TPM_ALG_ECDH, TPM_ALG_ECMQV, and TPM_ALG_SM2. In this procedure, the TPM performs the following steps: if the command is TPM2_Create(), it picks a random $a \in_R Z_q$ and computes $A = g^a$, and if the command is TPM2_CreatePrimary(), it derives a from a primary seed using a key derivation function and computes $A = g^a$; finally it returns A, and a key handle identifying a.[2]

First Phase The relevant command is TPM2_EC_Ephemeral(). This command is used to generate an ephemeral key. The TPM performs the following steps:

1. Generate $x = \mathsf{KDFa}(Random, Count)$, where KDFa() is a key derivation function described in [9], *Random* is a secure random value stored inside the TPM, and *Count* is a counter.
2. Set $ctr = Count$, $A[ctr] = 1$, $Count = Count + 1$, where $A[]$ is an array of bits used to indicate whether the ephemeral key has been used.
3. Set $x = x \bmod q$, and generate $X = g^x$.
4. Return X and ctr.

Note that the TPM doesn't need to store the ephemeral private key x as it can be recovered using KDFa() and ctr.

[2] Actually TPM2_Create() returns a key blob encrypted by a storage key, and the TPM2_Load() command loads the key blob and returns the key handle. For simplicity, we let TPM2_Create() directly return the key handle.

Second Phase The relevant command is TPM2_ZGen_2Phase(), which is the main command of tpm.KE. This command takes the following items as input:

scheme	a scheme selector indicating to the TPM which of the supported schemes is to be used
keyA	the key handle identifying the long-term private key a generated in the Key Generation procedure
ctr	the counter used to identify the ephemeral key generated in the first phase
B	the public key of $\hat{B}$, with which $\hat{A}$ wants to establish a session
Y	the ephemeral public key received from $\hat{B}$

1. The TPM first does the following checks:
   (a) Whether *scheme* equals the scheme designated for key A in the key generation procedure.
   (b) Whether B and Y are on the curve associated with A.
   (c) Whether $A[ctr] = 1$.
2. If the above checks succeed, the TPM recovers $x = \mathsf{KDFa}(Random, ctr)$, and performs the following steps:
   (a) Compute unhashed values according to the value of *scheme*:
      Case TPM_ALG_ECDH:
         set $Z_1 = B^a$, $Z_2 = Y^x$;
      Case TPM_ALG_ECMQV:
         set $Z_1 = (YB^e)^{h(x+da)}$, Z_2=NULL, where $d = avf(X)$ and $e = avf(Y)$;
      Case TPM_ALG_SM2:
         set $Z_1 = (BY^e)^{h(a+dx)}$, Z_2=NULL, where $d = avf'(X)$ and $e = avf'(Y)$;
   (b) Set $A[ctr] = 0$.
   (c) Return Z_1 and Z_2.
3. Finally, the host computes the session key after obtaining Z_1 and Z_2.

Note that when TPM2_ZGen_2Phase() completes successfully, the TPM clears $A[ctr]$, which ensures that the ephemeral private key x can only be used once.

3.2 Informal Analysis

We have shown that two weaknesses in the design of tpm.KE prevent it from achieving security property defined by modern AKE security models. One weakness is that tpm.KE returns Z_1 of the Full Unified Model protocol to the host whose memory is vulnerable to attacks, which makes Z_1 be available to the attacker. If the attacker obtains Z_1, the Full Unified Model protocol would be completely insecure as we have shown in Section 1.2. The other one is the weakness caused by the $avf()$ and $avf'()$, which results in group representation attacks on the MQV and SM2 key exchange protocols. Although this type of attacks is not feasible, it makes the two protocols cannot be proven secure.

We give two solutions to overcome the first weakness:

1. Perform the entire session key computation of Full Unified Model in the secure environment of the TPM, i.e., modify the TPM2_ZGen_2Phase() command not to return Z_1 and Z_2 but the session key, i.e., $K = H_1(Z_1, Z_2, \hat{A}, \hat{B}, X, Y)$.
2. Protect Z_1 and Z_2 from malicious code running on the host as much as possible such as keeping them only available in kernel mode, and delete Z_1 and Z_2 as soon as the session key is derived.

The first solution requires modifying the current TPM 2.0 specification, and the second one requires that the software code of session key derivation running on the host must be implemented properly and should be included in the Trusted Computing Base (TCB). The two solutions have the same purpose:

protecting Z_1 from the attacker, which helps us exclude Z_1 of Full Unified Model from the session state which the attacker can obtain in our formal analysis in Section 7.

As it seems that the second weakness only happens in theory, we perform a quantitative measure of the min-entropy contained in the output distribution of $avf()$ and $avf'()$ to check whether this weakness can happen in practice. We measure several series of widely deployed elliptic curves: the NIST series [12], the BN series [2], the SECG series [24], and an SM2 elliptic curve [1]. Our measure totals 17 elliptic curves and covers all elliptic curves adopted by TPM 2.0 [26]. We generate 16384 points for each elliptic curve, apply $avf'()$ to points of SM2 P256 curve, and apply $avf()$ to points of the rest curves[3]. We also apply the cryptographic hash function SHA-2 to the generated points of all curves. Then we measure the min-entropy of the output distributions of $avf()$ ($avf'()$) and SHA-2. The min-entropy results calculated using method of NIST SP 800-90 (formula 1) and CTW compression are summarized in Table 1. Table 1 also compares the min-entropy of the two output distributions. Figure 2 shows the development of the min-entropy value calculated using NIST's method over the number of measurements. To our surprise, the min-entropy of the output distribution of $avf()$ and $avf'()$ is very close to the min-entropy of the output distribution of SHA-2: the former is only about 1 bit less than the latter. What's more, the measure results indicate that the output distribution of $avf()$ and $avf'()$ is close to the uniform distribution. Take the measurement of BN P256 for example, the min-entropy calculated by the NIST's method is 126.93, very close to the output length of $avf()$ which is $129=\lceil 256/2 \rceil+1$, and the CTW ratio is 98.08% which is close to 1. Our practical measure indicates that the outputs of $avf()$ ($avf'()$) on different elliptic curve points are almost independent, and it is impractical for an attacker to mount group representation attacks on protocols based on practical elliptic curves. So in our formal analysis we model $avf()$ and $avf'()$ as random oracles.

Elliptic Curves		NIST 800-89		CTW Ratio	
		$avf()$	SHA-2	$avf()$	SHA-2
NIST Series	P192	95.19	95.94	97.13%	97.92%
	P224	111.01	111.99	97.68%	98.33%
	P256	126.86	127.89	98.08%	98.65%
	P384	190.19	191.30	98.95%	99.31%
	P521	258.73	259.80	100.01%	100.11%
BN Series	P192	95.09	96.15	97.13%	97.91%
	P224	111.03	111.95	97.67%	98.34%
	P256	126.93	127.95	98.08%	98.67%
	P384	190.23	191.19	98.95%	99.32%
	P512	253.62	254.80	99.35%	99.60%
	P638	314.97	316.13	100.04%	100.23%
SECG Series	P192	95.17	95.99	97.13%	97.92%
	P224	110.98	111.98	97.68%	98.34%
	P256	126.63	127.90	98.07%	98.65%
	P384	190.39	191.28	98.95%	99.31%
	P521	258.64	259.62	100.08%	100.11%
SM2	P256	125.81	126.89	100.14%	100.17%

Table 1. Min-entropy results

4 A Unified Security Model

This section presents our unified security model for tpm.KE, and describes the attacker model which models the capabilities of the attacker by some queries.

[3] $avf'()$ is defined only for SM2 key exchange, and $avf()$ is for MQV.

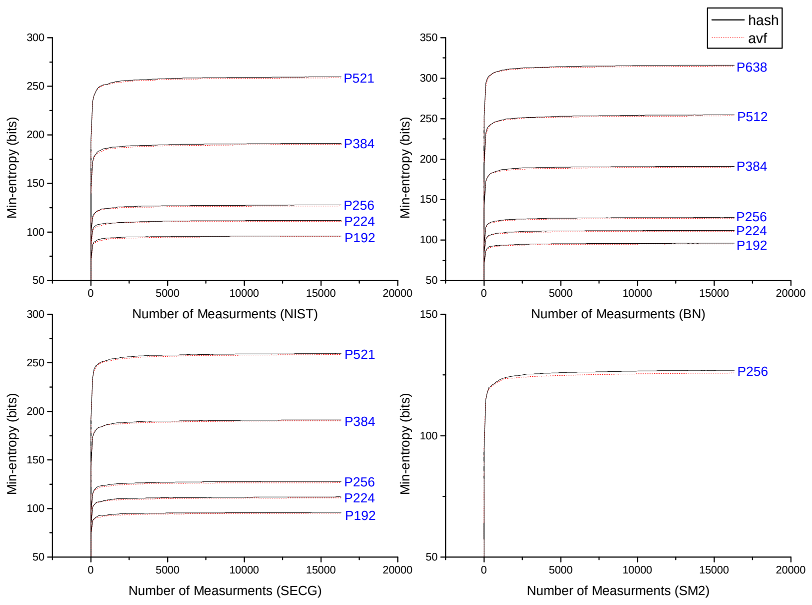

Fig. 2. Min-entropy evaluation

In our security model, each party has a long-term key generated by the TPM and a certificate (issued by a Certificate Authority (CA)) that binds the public key to the identity of that party. The long-term key can be one of the following three types: TPM_ALG_ECDH, TPM_ALG_ECMQV, and TPM_ALG_SM2. A party can be activated to invoke the interfaces of tpm.KE to run an instance of the protocol supported by the long-term key, and an instance of a protocol is called a session. In each session, a party can be activated as the role of initiator to send the first ephemeral public key or responder to send the second ephemeral public key by invoking the interface of the first phase of tpm.KE, and a party can complete the session by invoking the interface of the second phase of tpm.KE and computing the session key.

In previous AKE security models, a session is identified by a quadruple $(\hat{A}, \hat{B}, X, Y)$ where $\hat{A}$ is the identity of the owner of the session, $\hat{B}$ the peer party, X the outgoing ephemeral public key from $\hat{A}$, and Y the incoming ephemeral public key from $\hat{B}$. This kind of session identifier cannot identify a session established by tpm.KE as tpm.KE supports more than one scheme (protocol). So we use a quintuple $(sc, \hat{A}, \hat{B}, X, Y)$ to identify a session where sc denotes the scheme of the session. The session $(sc, \hat{B}, \hat{A}, Y, X)$ (if it exists) is said to be **matching** to session $(sc, \hat{A}, \hat{B}, X, Y)$, and the session $(sc', \hat{B}, \hat{A}, Y, X)$ where $sc' \neq sc$ (if it exists) is said to be **message-matching** to session $(sc, \hat{A}, \hat{B}, X, Y)$.

The introduction of sc to the session identifier brings an issue we must address: how about the security of the session $(sc, \hat{A}, \hat{B}, X, Y)$ if it has a corrupted message-matching session? Previous AKE security models don't capture this attack as they don't support formal analysis of multiple kinds of protocol in a unified way. However, this attack can happen on tpm.KE as it supports three AKE schemes and the TPM specification doesn't force the TPM to check the key type of its peer party. We say tpm.KE satisfies *correspondence property* if it can resist above attack, i.e., the session $(sc, \hat{A}, \hat{B}, X, Y)$ is secure if its message-matching session is compromised.

4.1 Attacker Model

The experiment involves multiple honest parties and an attacker $\mathcal{M}$ connected via an unauthenticated network. The attacker 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 attacker 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 tpm.KE and intermediate information stored and computed in the host.
– **SessionKeyReveal(s)**: $\mathcal{M}$ obtains the session key for the session s.
– **Corruption($\hat{P}$)**: In other AKE security models, this query allows $\mathcal{M}$ to learn the plaintext of the long-term private key of party $\hat{P}$. In our model, $\mathcal{M}$ doesn't learn anything about the plaintext of the private key but obtains the black-box access of the long-term key via TPM interfaces.
– **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 above three queries. A session is called *exposed* if $\mathcal{M}$ performs any one of above three queries to this session.

Note that our model differs from previous AKE security models in that the **Corruption** query to some party doesn't provide the attacker with the plaintext of the long-term private key of the party, but the black-box access of the long-term key which is randomly generated and protected by the TPM. This difference models the two protection features (see description in Section 1.2) provided by the TPM for tpm.KE.

The security is defined based on a game played by $\mathcal{M}$, in which $\mathcal{M}$ is allowed to activate sessions and perform SessionStateReveal, SessionKeyReveal, and Corruption 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 and its matching session (if it exists). Eventually $\mathcal{M}$ outputs a bit b' as its guess, then halts. $\mathcal{M}$ wins the game if $b' = b$. The attacker with above capabilities is called a **KE-attacker**. The formal security is defined as follows.

Definition 3. tpm.KE *is called secure if the following properties hold for any KE-attacker $\mathcal{M}$ 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.*

The first condition is a "consistency" requirement for sessions completed by two uncorrupted parties. The second condition is the core property for the security of tpm.KE: it guarantees that exposure of one session doesn't help the attacker to compromise the security of another session. Note that our security definition of tpm.KE allows the attacker to expose the message-matching session, that is to say, the test session is still secure even if the message-matching session is exposed by the attacker. Thus our model captures the correspondence property.

5 Formal Description of TPM.KE

This section formally describes tpm.KE from the view of how two-phase key exchange protocols can be implemented leveraging the TPM.

We use $\mathsf{ephem}_A()$ to model the interface of the first phase of $\mathsf{tpm.KE}$ where A is the long-term key of $\hat{A}$: once invoked, $\mathsf{ephem}_A()$ generates an ephemeral private/public key pair $(r, R = g^r)$, and returns an index *ctr* identifying the private key r in the TPM. We model as oracles the black-box manner of the key exchange functionalities provided by the second phase of $\mathsf{tpm.KE}$. The Full Unified Model, MQV, and SM2 key exchange functionalities provided by the second phase of $\mathsf{tpm.KE}$ are modeled as oracle $\mathcal{O}_A^{\mathsf{EC}}$, oracle $\mathcal{O}_A^{\mathsf{MQV}}$, and oracle $\mathcal{O}_A^{\mathsf{SM2}}$ respectively. $\mathcal{O}_A^{\mathsf{EC}}$ takes as input the input of $\mathsf{TPM2\_ZGen\_2Phase}()$, and returns the session key generated according to the specification of Full Unified Model. Note that we model our solutions to the first weakness of $\mathsf{tpm.KE}$ by letting $\mathcal{O}_A^{\mathsf{EC}}$ directly return the session key but not Z_1 and Z_2. $\mathcal{O}_A^{\mathsf{MQV}}$ and $\mathcal{O}_A^{\mathsf{SM2}}$ take as input the input of $\mathsf{TPM2\_ZGen\_2Phase}()$, and return the unhashed value according to specifications of the MQV and SM2 key exchange protocols respectively. We now formally describe $\mathsf{tpm.KE}$ by giving the following three session activations.

1. Initiate$(sc, \hat{A}, \hat{B})$: $\hat{A}$ invokes $\mathsf{ephem}_A()$ of its TPM to obtain an ephemeral public key X and an index ctr_x identifying the ephemeral private key x stored in the TPM, creates a local session which it identifies as (the incomplete) session $(sc, \hat{A}, \hat{B}, X)$ where sc is the key exchange scheme supported by the long-term key A, and outputs X as its outgoing ephemeral public key.
2. Respond$(sc, \hat{B}, \hat{A}, X)$ (sc is the scheme supported by B): After receiving X, $\hat{B}$ performs the following steps:
   (a) Invoke $\mathsf{ephem}_B()$ of its TPM to obtain an ephemeral public key Y and an index ctr_y identifying the ephemeral private key y stored in the TPM.
   (b) With input $(sc, keyB, ctr_y, A, X)$ where $keyB$ is the key handle of B, invoke corresponding oracle according to the value of sc:
      Case $\mathsf{TPM\_ALG\_ECDH}$: Invoke $\mathcal{O}_B^{\mathsf{EC}}$, set the session key K to be the return result of $\mathcal{O}_B^{\mathsf{EC}}$.
      Case $\mathsf{TPM\_ALG\_ECMQV}$: Invoke $\mathcal{O}_B^{\mathsf{MQV}}$, obtain Z_B from the return result, and compute the session key $K = H_2(Z_B, \hat{A}, \hat{B})$.
      Case $\mathsf{TPM\_ALG\_SM2}$: Invoke $\mathcal{O}_B^{\mathsf{SM2}}$, obtain Z_B from the return result, and compute the session key $K = H_2(Z_B, \hat{A}, \hat{B})$.
   (c) Complete the session with identifier $(sc, \hat{B}, \hat{A}, Y, X)$.
3. Complete$(sc, \hat{A}, \hat{B}, X, Y)$: $\hat{A}$ checks that it has an open session with identifier $(sc, \hat{A}, \hat{B}, X)$, then performs the following steps:
   (a) With input $(sc, keyA, ctr_x, B, Y)$ where $keyA$ is the key handle of A, invoke corresponding oracle according to the value of sc:
      Case $\mathsf{TPM\_ALG\_ECDH}$: Invoke $\mathcal{O}_A^{\mathsf{EC}}$, set the session key K to be the return result of $\mathcal{O}_A^{\mathsf{EC}}$.
      Case $\mathsf{TPM\_ALG\_ECMQV}$: Invoke $\mathcal{O}_A^{\mathsf{MQV}}$, obtain Z_A from the return result, and compute the session key $K = H_2(Z_A, \hat{A}, \hat{B})$.
      Case $\mathsf{TPM\_ALG\_SM2}$: Invoke $\mathcal{O}_A^{\mathsf{SM2}}$, obtain Z_A from the return result, and compute the session key $K = H_2(Z_A, \hat{A}, \hat{B})$.
   (b) Complete the session with identifier $(sc, \hat{A}, \hat{B}, X, Y)$.

6 Unforgeability of MQV and SM2 Key Exchange Functionalities

In this section, we first give formal definitions of MQV and SM2 Key Exchange functionalities provided by $\mathsf{tpm.KE}$, and formally prove their unforgeabilities with a constraint on the attacker. The unforgeabilities can simplify our formal analysis of $\mathsf{tpm.KE}$.

Definition 4 (MQV Functionality of $\mathsf{tpm.KE}$). *The functionality, denoted by $\mathcal{O}_B^{\mathsf{MQV}}$, is provided by a party possessing a private/public key pair $(b, B = g^b)$. A challenger, possessing a private/public key pair $(a, A = g^a)$, provides $\mathcal{O}_B^{\mathsf{MQV}}$ with a challenge $X = g^x$ (x is chosen and kept secret by the challenger). With the pair (A, X), $\mathcal{O}_B^{\mathsf{MQV}}$ first computes an ephemeral private/public key pair $(y, Y = g^y)$, and returns $Z = (XA^d)^{y+eb}$ where $d = avf(X)$ and $e = avf(Y)$. The challenger can verify the return result (Y, Z) with respect to challenge X by checking whether $Z = (YB^e)^{x+da}$.*

Definition 5 (SM2 Key Exchange Functionality of tpm.KE). *The functionality, denoted by $\mathcal{O}_B^{\mathsf{SM2}}$, is provided by a party possessing a private/public key pair $(b, B = g^b)$. A challenger, possessing a private/public key pair $(a, A = g^a)$, provides $\mathcal{O}_B^{\mathsf{SM2}}$ with a challenge $X = g^x$ (x is chosen and kept secret by the challenger). With the pair (A, X), $\mathcal{O}_B^{\mathsf{SM2}}$ first computes an ephemeral private/public key pair $(y, Y = g^y)$, and returns $Z = (AX^d)^{b+ey}$ where $d = avf'(X)$ and $e = avf'(Y)$. The challenger can verify the return result (Y, Z) with respect to challenge X by checking whether $Z = (BY^e)^{a+dx}$.*

Theorem 1. *Under the CDH assumption, with $avf()$ modeled as a random oracle, given a challenge X, it is computationally infeasible for an attacker to forge a return result of $\mathcal{O}_B^{\mathsf{MQV}}$ on behalf of a challenger whose public key is A under the constraint that (a, x) is unknown to the attacker.*

It's straight to see that if (a, x) is known to an attacker, it's feasible for him to forge a return result of $\mathcal{O}_B^{\mathsf{MQV}}$ by computing $(YB^e)^{x+da}$ where Y is randomly generated by himself. That's why we add the constraint that (a, x) is unknown to the attacker. We prove Theorem 1 by showing that if an attacker $\mathcal{M}$ can forge a return result under our constraint, then we can construct a CDH solver $\mathcal{C}$ which uses $\mathcal{M}$ as a subroutine.

Proof. $\mathcal{C}$ takes as input a pair $(X, B) \in G^2$ and $(a, A = g^a)$, and simulates $\mathcal{O}_B^{\mathsf{MQV}}$ as follows:

1. On receipt input (A', X'), choose $e, s \in Z_q$ randomly.
2. Let $Y' = g^s/B^e$, and set $avf(Y') = e$.
3. Choose d randomly, and set $avf(X') = d$.
4. Return $(Y, Z' = (X'A^d)^s)$.

If $\mathcal{M}$ successfully forges a return result (Y, Z) on the pair (A, X) in an experiment, then $\mathcal{C}$ obtains $Z = (XA^d)^{y+eb}$ where $d = avf(X)$ and $e = avf(Y)$. Note that without the knowledge of private key y of Y, $\mathcal{C}$ is unable to compute CDH(X, B). Following the Forking Lemma [23] approach, $\mathcal{C}$ runs $\mathcal{M}$ on the same input and the same coin flips but with carefully modified answers to $avf()$ queries. Note that $\mathcal{M}$ must have queried $avf(Y)$ in its first run, because otherwise $\mathcal{M}$ would be unable to compute Z. For the second run of $\mathcal{M}$, $\mathcal{C}$ responds to $avf(Y)$ with a value $e' \neq e$ selected uniformly at random. If $\mathcal{M}$ succeeds in the second run, $\mathcal{C}$ obtains $Z' = (XA^d)^{y+e'b}$, and therefore can compute CDH$(X, B) = (\frac{Z}{Z'})^{\frac{1}{e-e'}} B^{-da}$. □

Theorem 2. *Under the CDH assumption, with $avf'()$ modeled as a random oracle, given a challenge X, it is computationally infeasible for an attacker to forge a return result of $\mathcal{O}_B^{\mathsf{SM2}}$ on behalf of a challenger whose public key is A under the constraint that (a, x) is unknown to the attacker.*

We omit the proof of theorem 2 as it can be easily completed following the proof of theorem 1.

7 Security Analysis of tpm.KE

In this section, we analyze the security of tpm.KE in the security model defined in Section 4. We first define the session state allowed to be revealed by the attacker.

Session State. In order to simulate the protections provided by the TPM, we specify that a session state stores the results returned by the TPM and the information stored in the host. For the Full Unified Model scheme, the session state is the session key; for the MQV and SM2 key exchange schemes, the session state is the unhashed value returned by the TPM.

Theorem 3. *Under the CDH and GDH assumptions, with hash functions $H_1()$ and $H_2()$, $avf()$, and $avf'()$ modeled as random oracles,* tpm.KE *is secure in our unified model.*

The proof of above theorem follows from the definition of secure key exchange protocols outlined in Section 4 and the following two lemmas.

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

Lemma 2. *Under the CDH and GDH assumptions, there is no feasible attacker 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 $(sc, \hat{A}, \hat{B}, X, Y)$ and $\hat{B}$ completes the matching session $(sc, \hat{B}, \hat{A}, Y, X)$, then it's easy to verify that the session key computed by $\hat{A}$ is the same as the session key computed by $\hat{B}$ according to specifications of the protocols described in Figure 1.

The rest of this section proves Lemma 2. Let $\mathcal{M}$ be any attacker against tpm.KE. We observe that the session key of the test session is computed as: (1) $K = H_1(\sigma)$ for some 4-tuple σ if the TPM_ALG_ECDH scheme is selected; (2) $K = H_2(\sigma)$ for some 3-tuple σ if the TPM_ALG_ECMQV or the TPM_ALG_SM2 scheme is selected. The attacker $\mathcal{M}$ has only two ways to distinguish K from a random value:

1. Forging attack. At some point $\mathcal{M}$ queries $H_1()$ or $H_2()$ on the same tuple σ as 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.

We will show that if either of the attacks succeeds with non-negligible probability then there exists an attacker against the GDH problem, or a forger against the MQV Functionality of tpm.KE, or a forger against the SM2 key exchange Functionality of tpm.KE. The latter two forgers are in contradiction to the CDH assumption (theorem 1 and theorem 2).

7.1 Infeasibility of Forging Attacks

Consider a successful attack performed by $\mathcal{M}$. Let $(sc, \hat{A}, \hat{B}, X_0, Y_0)$ be the test session for which $\mathcal{M}$ outputs a correct guess for the tuple 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 $\hat{A}$. sc can fall under one of the following three cases:

1. $sc =$ TPM_ALG_ECDH.
2. $sc =$ TPM_ALG_ECMQV.
3. $sc =$ TPM_ALG_SM2.

As we assume that $\mathcal{M}$ succeeds with non-negligible probability in the forging attack then there is at least one of the above three cases that occurs with non-negligible probability. We assume that $\mathcal{M}$ operates in an environment that involves at most n parties and each party participates in at most k sessions. We proceed to analyze these cases separately.

7.1.1 Analysis of Case 1

For this case, we build a GDH solver $\mathcal{S}_1$ with the following property: if $\mathcal{M}$ succeeds with non-negligible probability in this case, then $\mathcal{S}_1$ succeeds with non-negligible probability in solving the GDH problem. $\mathcal{S}_1$ takes as input a pair (A, B), creates an experiment which includes n honest parties and the attacker $\mathcal{M}$, and is given access to a DDH oracle DDH. $\mathcal{S}_1$ randomly selects two party $\hat{A}$ and $\hat{B}$ from the honest parties and sets their public keys to be A and B respectively. All the other parties compute their keys normally. Furthermore, $\mathcal{S}_1$ randomly selects an integer $i \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows:

0. $\mathcal{S}_1$ models $\mathsf{ephem}_P()$ for all parties except party $\hat{B}$ following the description in Section 5. $\mathcal{M}$ sets the type of all long-term keys. If the type of A is not $\mathsf{TPM\_ALG\_ECDH}$, $\mathcal{S}_1$ aborts. $\mathcal{S}_1$ creates oracles modeling the two-phase key exchange functionalities for each party normally except the oracles for parties $\hat{A}$ and $\hat{B}$ as it possesses the long-term private key of all parties except $\hat{A}$ and $\hat{B}$. $H_1()$, $H_2()$, $avf()$ and $avf'()$ are modeled as random oracles described below.
1. Initiate$(sc, \hat{P}_1, \hat{P}_2)$: $\hat{P}_1$ executes the Initiate() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}_1$ checks whether $sc = \mathsf{TPM\_ALG\_ECDH}$ and $\hat{P}_2$ is $\hat{B}$. If not, $\mathcal{S}_1$ aborts.
2. Respond$(sc, \hat{P}_1, \hat{P}_2, Y)$: With the exception of $\hat{A}$ and $\hat{B}$ (whose behaviors 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 $\hat{A}$, $\mathcal{S}_1$ checks whether $sc = \mathsf{TPM\_ALG\_ECDH}$ and $\hat{P}_2$ is $\hat{B}$. If not, $\mathcal{S}_1$ aborts.
3. Complete$(sc, \hat{P}_1, \hat{P}_2, X, Y)$: With the exception of $\hat{A}$ and $\hat{B}$ (whose behaviors we explain below), $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session is the i-th session at $\hat{A}$, $\mathcal{S}_1$ completes the session without computing a session key.
4. With the input $(sc, keyA, ctr_x, P, Y)$, $\mathcal{S}_1$ creates the oracle $\mathcal{O}_A^{\mathsf{EC}}$ as follows:
   (a) If $\hat{P} = \hat{B}$, $\mathcal{O}_A^{\mathsf{EC}}$ returns a session key to be $H_{spec}(\hat{A}, \hat{B}, X, Y)$. $H_{spec}()$ is simulated as a random oracle.
   (b) If $\hat{P} \neq \hat{B}$, returns a session key to be $H_1(Z_1, Z_2, \hat{A}, \hat{B}, X, Y)$ where $Z_1 = A^p$ (p is the long-term private key of party $\hat{P}$) and $Z_2 = Y^x$ (x is the ephemeral private key indexed by ctr_x).
5. Now $\mathcal{S}_1$ can simulate all the session activations at $\hat{A}$ for $\mathcal{M}$ with the help of $\mathcal{O}_A^{\mathsf{EC}}$.
6. $\mathcal{S}_1$ creates a Table T and models $\mathsf{ephem}_B()$ according to the type of B:
   (a) Case $\mathsf{TPM\_ALG\_ECDH}$: Model following the description in Section 5.
   (b) Case $\mathsf{TPM\_ALG\_ECMQV}$:
      i. Choose $e, s \in Z_q$ randomly.
      ii. Set $Y = g^s/B^e$ and $e = avf(Y)$.
      iii. Randomly choose an index ctr, and add a record $(ctr, e, s, Y, -)$ to T.
      iv. Return ctr and Y.
   (c) Case $\mathsf{TPM\_ALG\_SM2}$:
      i. Choose $e, s \in Z_q$ randomly.
      ii. Set $Y = (g^s/B)^{e^{-1}}$ and $e = avf'(Y)$.
      iii. Randomly choose an index ctr, and add a record $(ctr, e, s, Y, -)$ to T.
      iv. Return ctr and Y.
7. With the input $(sc, keyB, ctr_y, P, X)$, $\mathcal{S}_1$ creates the oracle modeling the two-phase key exchange functionality for party $\hat{B}$ according to the type of B:
   (a) Case $\mathsf{TPM\_ALG\_ECDH}$: $\mathcal{O}_B^{\mathsf{EC}}$ is modeled similarly to $\mathcal{O}_A^{\mathsf{EC}}$ which is described in step 4.
   (b) Case $\mathsf{TPM\_ALG\_ECMQV}$:
      i. Check whether (1) $sc = \mathsf{TPM\_ALG\_ECMQV}$, and (2) P and X are on the curve associated with B, and (3) the last element of the record in T indexed by ctr_y is '-'. If above checks succeed, continue, else return error.
      ii. Suppose the record in T indexed by ctr_y is $(ctr, e, s, Y, -)$, set $Z_1 = (XP^d)^s$ where $d = avf(X)$.
      iii. Return (Z_1, Z_2=NULL).
   (c) Case $\mathsf{TPM\_ALG\_SM2}$:
      i. Check whether (1) $sc = \mathsf{TPM\_ALG\_SM2}$, and (2) P and X are on the curve associated with B, and (3) the last element of the record in T indexed by ctr_y is '-'. If above checks pass, continue, else return error.
      ii. Suppose the record in T indexed by ctr_y is $(ctr, e, s, Y, -)$, set $Z_1 = (PX^d)^s$ where $d = avf'(X)$.
      iii. Return (Z_1, Z_2=NULL).

8. $\mathcal{S}_1$ simulates all the session activations at $\hat{B}$ for $\mathcal{M}$ with the help of $\mathsf{ephem}_B()$ and the oracle created in step 7.
9. SessionStateReveal(s): $\mathcal{S}_1$ returns to $\mathcal{M}$ the session state of session s. However, if s is the i-th session at $\hat{A}$, $\mathcal{S}_1$ aborts.
10. SessionKeyReveal(s): $\mathcal{S}_1$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$, $\mathcal{S}_1$ aborts.
11. Corruption($\hat{P}$): $\mathcal{S}_1$ gives $\mathcal{M}$ the handle of the long-term key P. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}_1$ aborts.
12. $H_1(\sigma)$ function for some $\sigma = (Z_1, Z_2, \hat{P}_1, \hat{P}_2, X, Y)$ proceeds as follows:
    (a) If $\hat{P}_1 = \hat{A}$, $\hat{P}_2 = \hat{B}$, and $DDH(A, B, Z_1) = 1$, then $\mathcal{S}_1$ aborts $\mathcal{M}$ and is successful by outputting CDH(A, B)=Z_1.
    (b) If the value of the function on input σ has been previously defined, return it.
    (c) If the value of $H_{spec}()$ on input $(\hat{P}_1, \hat{P}_2, X, Y)$ has been previously defined, return it.
    (d) Pick a key k randomly from the key distribution, and define $H_1(\sigma) = k$
13. $H_{spec}()$, $H_2()$, $avf()$, and $avf'()$ are simulated as random oracles in the usual way.

Proof. The probability that $\mathcal{M}$ sets the type of A to be TPM_ALG_ECDH, and selects the i-th session of $\hat{A}$ and the peer of the test session is party $\hat{B}$ is at least $\frac{1}{3n^2k}$. Suppose that this indeed the case: the type of A is TPM_ALG_ECDH, so $\mathcal{S}_1$ doesn't abort in Step 0; $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$, so $\mathcal{S}_1$ doesn't abort in Step 1, 2, 9, 10, 11. So $\mathcal{S}_1$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. Thus, if $\mathcal{M}$ wins with non-negligible probability in this case, the success probability of $\mathcal{S}_1$ is bounded by:

$$Pr(\mathcal{S}_1) \geq \frac{1}{3n^2k} Pr(\mathcal{M}).$$

□

7.1.2 Analysis of Case 2

Recall that the test session is denoted by $(sc, \hat{A}, \hat{B}, X_0, Y_0)$. We divide case 2 of the forging attack into the following four subcases according to the generation of Y_0:

C1. Y_0 was generated by $\hat{B}$ in a session matching the test session, i.e., in session $(sc, \hat{B}, \hat{A}, Y_0, X_0)$.
C2. Y_0 was generated by $\hat{B}$ in a session message-matching the test session, i.e., in session $(sc', \hat{B}, \hat{A}, Y_0, X_0)$ with $sc' \neq sc$.
C3. Y_0 was generated by $\hat{B}$ in a session $(sc', \hat{B}, \hat{A}^*, Y_0, X^*)$ with $(\hat{A}*, X^*) \neq (\hat{A}, X_0)$.
C4. Y_0 didn't appear in any completed sessions activated at $\hat{B}$, i.e., Y_0 was never output by $\hat{B}$ as its outgoing ephemeral public key in any sessions or $\hat{B}$ did output Y_0 as its outgoing ephemeral public key for some session s but it never completed s by computing the session key.

If $\mathcal{M}$ succeeds in Case 2 in its forging attack with non-negligible probability then there is at least one of the above 4 subcases happens with non-negligible probability in the successful runs of $\mathcal{M}$. We proceed to analyze these subcases separately.
Analysis of Subcase C1. For this subcase, we build a GDH solver $\mathcal{S}_2$ with the following property: if $\mathcal{M}$ succeeds with non-negligible probability in this subcase, then $\mathcal{S}_2$ succeeds with non-negligible probability in solving the GDH problem. $\mathcal{S}_2$ takes as input a pair (X_0, Y_0), creates an experiment which includes n honest parties and the attacker $\mathcal{M}$, and is given access to a DDH oracle DDH. All parties compute their keys normally. $\mathcal{S}_2$ randomly selects two party $\hat{A}$ and $\hat{B}$, and randomly selects two integers $i, j \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows:

0. $\mathcal{S}_2$ models $\mathsf{ephem}_P()$ for all parties following the description in Section 5. $\mathcal{M}$ sets the type of all long-term keys. If the type of A and B is not TPM_ALG_ECMQV, $\mathcal{S}_2$ aborts. $\mathcal{S}_2$ can create oracles modeling the two-phase key exchange functionalities for each party normally as it possesses the long-term private key of all parties.

1. Initiate($sc, \hat{P}_1, \hat{P}_2$): $\hat{P}_1$ 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 $\hat{B}$), $\mathcal{S}_2$ checks whether $sc =$ TPM_ALG_ECMQV and $\hat{P}_2$ is $\hat{B}$ (or $\hat{P}_2$ is $\hat{A}$). If so, $\mathcal{S}_2$ sets the ephemeral public key to be X_0 (or Y_0), else $\mathcal{S}_2$ aborts.
2. Respond($sc, \hat{P}_1, \hat{P}_2, Y$): $\hat{P}_1$ 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 $\hat{B}$), $\mathcal{S}_2$ checks whether $sc =$ TPM_ALG_ECMQV, $\hat{P}_2$ is $\hat{B}$ (or $\hat{P}_2$ is $\hat{A}$), and $Y = Y_0$ (or $Y = X_0$). If so, $\mathcal{S}_2$ sets the ephemeral public key to be X_0 (or Y_0), else $\mathcal{S}_2$ aborts.
3. Complete($sc, \hat{P}_1, \hat{P}_2, X, Y$): $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}_2$ completes the session without computing a session key.
4. SessionStateReveal(s): $\mathcal{S}_2$ returns to $\mathcal{M}$ the session state of session s. However, if s is the i-th session at $\hat{A}$ (or the j-th session at $\hat{B}$), $\mathcal{S}_2$ aborts.
5. SessionKeyReveal(s): $\mathcal{S}_2$ 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 $\hat{B}$), $\mathcal{S}_2$ aborts.
6. Corruption($\hat{P}$): $\mathcal{S}_2$ gives $\mathcal{M}$ the handle of the long-term key P. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}_2$ aborts.
7. $H_2(\sigma)$ function for some $\sigma = (Z, \hat{P}_1, \hat{P}_2)$ proceeds as follows:
   (a) If $\hat{P}_1 = \hat{A}$, $\hat{P}_2 = \hat{B}$, and $DDH(X_0A^d, Y_0B^e, Z) = 1$ where $d = avf(X_0)$ and $e = avf(Y_0)$, then $\mathcal{S}_2$ aborts $\mathcal{M}$ and is successful by outputting CDH$(X_0, Y_0) = \frac{Z}{X_0^{eb} Y_0^{da} g^{deab}}$.
   (b) If the value of the function on input σ has been previously defined, return it.
   (c) Pick a key k randomly from the key distribution, and define $H_2(\sigma) = k$
8. $H_1()$, $avf()$, and $avf'()$ are simulated as random oracles in the usual way.

Proof. The probability that $\mathcal{M}$ sets the type of A and B to be TPM_ALG_ECMQV, and 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 $\frac{1}{3} \times \frac{1}{3} \times \frac{2}{(nk)^2} = \frac{2}{9(nk)^2}$. Suppose that this is indeed the case: the type of A and B is TPM_ALG_ECMQV, so $\mathcal{S}_2$ doesn't abort in Step 0; $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$ or the j-th session of $\hat{B}$, so $\mathcal{S}_2$ doesn't abort in Step 1, 2, 4, 5, 6. So $\mathcal{S}_2$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. Thus, if $\mathcal{M}$ wins with non-negligible probability in this case, the success probability of $\mathcal{S}_2$ is bounded by:

$$Pr(\mathcal{S}_2) \geq \frac{2}{9(nk)^2} Pr(\mathcal{M}).$$

□

Analysis of Subcase C2. For this subcase, we show that $\mathcal{M}$ can break the unforgeability of the MQV functionality proved in theorem 1 if it succeeds with non-negligible probability. We build a simulator $\mathcal{S}_3$ which simulates $\mathcal{M}$'s environment. $\mathcal{S}_3$ takes as input a challenge X_0, creates an experiment which includes n honest parties and the attacker $\mathcal{M}$. All parties compute their keys normally. $\mathcal{S}_3$ randomly selects two party $\hat{A}$ and $\hat{B}$, and randomly selects two integers $i, j \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows:

0. $\mathcal{S}_3$ models ephem$_P$() for all parties following the description in Section 5. $\mathcal{M}$ sets the type for all long-term keys. If the type of A is not TPM_ALG_ECMQV and the type of B is TPM_ALG_ECMQV, $\mathcal{S}_3$ aborts. $\mathcal{S}_3$ can create oracles modeling the two-phase key exchange functionalities for each party normally as it possesses the long-term private key of all parties.
1. Initiate($sc, \hat{P}_1, \hat{P}_2$): $\hat{P}_1$ executes the Initiate() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}_3$ checks whether $sc =$ TPM_ALG_ECMQV and $\hat{P}_2$ is $\hat{B}$. If so, $\mathcal{S}_3$ sets the ephemeral public key to be X_0, else $\mathcal{S}_3$ aborts. If the session being created is the j-th session at $\hat{B}$, $\mathcal{S}_3$ checks whether $sc \neq$ TPM_ALG_ECMQV and $\hat{P}_2$ is $\hat{A}$. If so, $\mathcal{S}_3$ calls ephem$_B$() to create an ephemeral key, denoted by Y_0, and sets the outgoing ephemeral key of this session to be Y_0, else $\mathcal{S}_3$ aborts.

2. Respond($sc, \hat{P}_1, \hat{P}_2, Y$): $\hat{P}_1$ executes the Respond() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}_3$ checks whether $sc =$ TPM_ALG_ECMQV, $\hat{P}_2$ is $\hat{B}$, and $Y = Y_0$. If so, $\mathcal{S}_3$ provides $\mathcal{M}$ with the value X_0, else $\mathcal{S}_3$ aborts. If the session being created is the j-th session at $\hat{B}$, $\mathcal{S}_3$ checks whether $sc \neq$ TPM_ALG_ECMQV, $\hat{P}_2$ is $\hat{A}$, and $Y = X_0$. If so, $\mathcal{S}_3$ calls $\mathsf{ephem}_B()$ to create an ephemeral key, denoted by Y_0, and sets the outgoing ephemeral key of this session to be Y_0, else $\mathcal{S}_3$ aborts.
3. Complete($sc, \hat{P}_1, \hat{P}_2, X, Y$): $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session is the i-th session at $\hat{A}$, $\mathcal{S}_3$ completes the session without computing a session key.
4. SessionStateReveal(s): $\mathcal{S}_3$ returns to $\mathcal{M}$ the session state of session s. However, if s is the i-th session at $\hat{A}$, $\mathcal{S}_3$ aborts.
5. SessionKeyReveal(s): $\mathcal{S}_3$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$, $\mathcal{S}_3$ aborts.
6. Corruption($\hat{P}$): $\mathcal{S}_3$ gives $\mathcal{M}$ the handle of the long-term key P. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}_3$ aborts.
7. $H_1()$, $H_2()$, $avf()$, and $avf'()$ are simulated as random oracles in the usual way.

Proof. The probability that $\mathcal{M}$ sets the type of A to be TPM_ALG_ECMQV, and the type of B not to be TPM_ALG_ECMQV, and selects the i-th session of $\hat{A}$ and the j-th session of $\hat{B}$ as the test session and its message-matching session is at least $\frac{1}{3} \times \frac{2}{3} \times \frac{1}{(nk)^2} = \frac{2}{9(nk)^2}$. Suppose that this is indeed the case: the type of A is TPM_ALG_ECMQV and the type of B is not TPM_ALG_ECMQV, so $\mathcal{S}_3$ doesn't abort in Step 0; $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$, so $\mathcal{S}_3$ doesn't abort in Step 1, 2, 4, 5, 6. So $\mathcal{S}_2$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. By the assumption, $\mathcal{M}$ correctly guesses the tuple $(Z = (Y_0B^e)^{x_0+da}, \hat{A}, \hat{B})$ of the test session where $d = avf(X_0)$ and $e = avf(Y_0)$. We now show that (Y_0, Z) is a valid forgery against $\mathcal{O}_B^{\mathsf{MQV}}$ on input (X_0, A) where X_0 is the challenge:

1. (Y_0, Z) is a valid return result of $\mathcal{O}_B^{\mathsf{MQV}}$ as $Z = (Y_0B^e)^{x_0+da} = (X_0A^d)^{y_0+eb}$.
2. $\mathcal{O}_B^{\mathsf{MQV}}$ never returns the result (Y_0, Z) on input (X_0, A) under $\mathcal{S}_3$: $\mathcal{O}_B^{\mathsf{MQV}}$ has never been created by $\mathcal{S}_3$ as the type of B is not TPM_ALG_ECMQV in this subcase.
3. Since $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$, $\mathcal{M}$ doesn't know a and b. So $\mathcal{M}$ doesn't know the private key pair (a, x_0). Thus, $\mathcal{M}$ is under the constraint described in theorem 1.

Finally we get: $Pr(\mathcal{M}$ succeeds in forging $\mathcal{O}_B^{\mathsf{MQV}}$ under $\mathcal{S}_3) \geq \frac{2}{9(nk)^2} Pr(\mathcal{M})$. □

Analysis of Subcases C3 and C4. For the two subcases, we show that $\mathcal{M}$ can break the unforgeability of the MQV functionality proved in theorem 1 if it succeeds with non-negligible probability. We build a simulator $\mathcal{S}_4$ which simulates $\mathcal{M}$'s environment. $\mathcal{S}_4$ takes as input a challenge X_0, creates an experiment which includes n honest parties and the attacker $\mathcal{M}$. All parties compute their keys normally. $\mathcal{S}_4$ randomly selects two party $\hat{A}$ and $\hat{B}$, and randomly selects one integer $i \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows:

0. $\mathcal{S}_4$ models $\mathsf{ephem}_P()$ for all parties following the description in Section 5. $\mathcal{M}$ sets the type for all long-term keys. If the type of A is not TPM_ALG_ECMQV, $\mathcal{S}_3$ aborts. $\mathcal{S}_4$ can create oracles modeling the two-phase key exchange functionalities for each party normally as it possesses the long-term private key of all the parties.
1. Initiate($sc, \hat{P}_1, \hat{P}_2$): $\hat{P}_1$ executes the Initiate() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}_4$ checks whether $sc =$ TPM_ALG_ECMQV and $\hat{P}_2$ is $\hat{B}$. If so, $\mathcal{S}_4$ sets the ephemeral public key to be X_0, else $\mathcal{S}_4$ aborts.
2. Respond($sc, \hat{P}_1, \hat{P}_2, Y$): $\hat{P}_1$ executes the Respond() activation of the protocol. However, if the session being created is the i-th session at $\hat{A}$, $\mathcal{S}_4$ checks whether $sc =$ TPM_ALG_ECMQV and $\hat{P}_2$ is $\hat{B}$. If so, $\mathcal{S}_4$ provides $\mathcal{M}$ with the value X_0, else $\mathcal{S}_4$ aborts.

3. Complete($sc, \hat{P}_1, \hat{P}_2, X, Y$): $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session is the i-th session at $\hat{A}$, $\mathcal{S}_4$ completes the session without computing a session key.
4. SessionStateReveal(s): $\mathcal{S}_4$ returns to $\mathcal{M}$ the session state of session s. However, if s is the i-th session at $\hat{A}$, $\mathcal{S}_4$ aborts.
5. SessionKeyReveal(s): $\mathcal{S}_4$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$, $\mathcal{S}_4$ aborts.
6. Corruption($\hat{P}$): $\mathcal{S}_4$ gives $\mathcal{M}$ the handle of the long-term key P. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}_4$ aborts.
7. $H_1()$, $H_2()$, $avf()$, and $avf'()$ are simulated as random oracles in the usual way.

Proof. The probability that $\mathcal{M}$ sets the type of A to be $\mathsf{TPM\_ALG\_ECMQV}$, and selects the i-th session of $\hat{A}$ as the test session is at least $\frac{1}{3} \times \frac{1}{n^2k} = \frac{1}{3n^2k}$. Suppose that this is indeed the case: the type of A is $\mathsf{TPM\_ALG\_ECMQV}$, so $\mathcal{S}_4$ doesn't abort in Step 0; $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$, make SessionStateReveal and SessionKeyReveal queries to the i-th session of $\hat{A}$, so $\mathcal{S}_4$ doesn't abort in Step 1, 2, 4, 5, 6. So $\mathcal{S}_4$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. By the assumption, $\mathcal{M}$ correctly guesses the tuple $(Z = (Y_0B^e)^{x_0+da}, \hat{A}, \hat{B})$ of the test session where $d = avf(X_0)$ and $e = avf(Y_0)$. We now show that (Y_0, Z) is a valid forgery against $\mathcal{O}_B^{\mathsf{MQV}}$ on input (X_0, A) where X_0 is the challenge:

1. (Y_0, Z) is a valid return result of $\mathcal{O}_B^{\mathsf{MQV}}$ as $Z = (Y_0B^e)^{x_0+da} = (X_0A^d)^{y_0+eb}$.
2. We now show that $\mathcal{O}_B^{\mathsf{MQV}}$ never returns the result (Y_0, Z) on input (X_0, A) under $\mathcal{S}_4$:
   (a) If the type of B is $\mathsf{TPM\_ALG\_ECDH}$ or $\mathsf{TPM\_ALG\_SM2}$, then $\mathcal{O}_B^{\mathsf{MQV}}$ has never been created by $\mathcal{S}_4$.
   (b) If the type of B is $\mathsf{TPM\_ALG\_ECMQV}$, then $\mathcal{S}_4$ must create $\mathcal{O}_B^{\mathsf{MQV}}$ in step 0. However, if $\mathcal{O}_B^{\mathsf{MQV}}$ ever returned the result (Y_0, Z) for some Z on input (A, X_0), then $\hat{B}$ must have an session which is identified by $(sc = \mathsf{TPM\_ALG\_ECMQV}, \hat{B}, \hat{A}, Y_0, X_0)$, which is exactly the matching session of the test session. This contradicts that the test session has no matching session in these two subcases.
3. Since $\mathcal{M}$ is not allowed to corrupt $\hat{A}$ and $\hat{B}$, $\mathcal{M}$ doesn't know a and b. So $\mathcal{M}$ doesn't know the private key pair (a, x_0). Thus, $\mathcal{M}$ is under the constraint described in theorem 1.

Finally we get: $Pr(\mathcal{M}$ succeeds in forging $\mathcal{O}_B^{\mathsf{MQV}}$ under $\mathcal{S}_4) \geq \frac{1}{3n^2k} Pr(\mathcal{M})$. □

7.1.3 Analysis of Case 3

The analysis of Case 3 is very similar to Case 2. It is easy to get a full proof by following the analysis from Section 7.1.2, so we omit the analysis of Case 3.

7.2 Infeasibility of Key-replication Attacks

Consider a successful key-replication attack performed by $\mathcal{M}$ against the test session $s = (sc, \hat{A}, \hat{B}, X_0, Y_0)$. That is to say, $\mathcal{M}$ succeeds in establishing a session $s' = (sc', \hat{A}', \hat{B}', X', Y')$, which is different than s and $(sc, \hat{B}, \hat{A}, Y_0, X_0)$ (the matching session of s) but has the same key as the test session. sc can fall under one of the following cases:

1. $sc = \mathsf{TPM\_ALG\_ECDH}$.
2. $sc = \mathsf{TPM\_ALG\_ECMQV}$.
3. $sc = \mathsf{TPM\_ALG\_SM2}$.

Since we assume that $\mathcal{M}$ succeeds with non-negligible probability in the key-replication attack then there is at least one of the above three cases that occurs with non-negligible probability. We proceed to analyze these cases separately.

7.2.1 Analysis of Case 1

We show that a key-replication attack is impossible in this case. Note that in this case the session key of the test session is the value of the random oracle $H_1()$ on $\sigma = (Z_1, Z_2, \hat{A}, \hat{B}, X_0, Y_0)$. As the session key of the MQV or SM2 key exchange protocol is the value of the random oracle $H_2()$ on some 3-tuple $(Z, \hat{A}, \hat{B})$, the session s' must belong to a party whose long-term key is the type of $\mathsf{TPM\_ALG\_ECDH}$. So the scheme of s' must be $\mathsf{TPM\_ALG\_ECDH}$. This means that the session identifier of s' must be $(sc, \hat{A}, \hat{B}, X_0, Y_0)$ or $(sc, \hat{B}, \hat{A}, Y_0, X_0)$ where $sc = \mathsf{TPM\_ALG\_ECDH}$, i.e., s' is the test session or its matching session, which contradicts that s' is different from s and the matching session of s.

7.2.2 Analysis of Cases 2 and 3

We show that a key-replication attack is impossible by showing that a successful attacker would contradict the GDH assumption, or break the unforgeabilities of MQV functionality or SM2 key exchange functionality.

Consider the simulators $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ built above for the four subcases of Case 2 in Section 7.1.2. In these subcases, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ provide $\mathcal{M}$ with the session state of all exposed sessions. Note that the session key of the test session is the value of the random oracle $H_2(Z, \hat{A}, \hat{B})$. So s' must have the same σ as the test session. Therefore, if $\mathcal{M}$ is able to succeed in a key-replication attack against the test session s, then it can obtain the 3-tuples of s by exposing s' (this is allowed in the security model as s' is not the matching session of s). This means that $\mathcal{M}$ is able to launch forging attacks. However, we have shown that if $\mathcal{M}$ succeeds in a forging attack: $\mathcal{S}_2$ would succeed in solving the GDH problem, and under $\mathcal{S}_3$ and $\mathcal{S}_4$ there would exist an attacker breaking the unforgeability of the MQV functionality of tpm.KE. By applying above argument and replacing the unforgeability of the MQV functionality of tpm.KE with the unforgeability of the SM2 key exchange functionality of tpm.KE, we can get the analysis of Case 3.

8 Discussion and Suggestions

In this section, we first discuss some further security properties for AKE protocols, then give suggestions on how to use tpm.KE securely and suggestions on how to improve the security of tpm.KE.

8.1 Further Security Properties

Besides the basic security property defined by modern security models, it's desirable for AKE protocols 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 enable the attacker 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.

Note that our security model doesn't capture the KCI resistance property and PFS property as our model doesn't allow the attacker to obtain the plaintext of the long-term private key but only allows the attacker to control the handle of the long-term key. The reason that we put such constraint on the attacker, which is used to model protections provided by the TPM hardware chip, is that we aim to check whether the tpm.KE defined by current TPM specification can provide a secure key exchange functionality (Note that in scenarios where long-term keys can be obtained by the attacker, for example keys are not protected by hardware tokens, all the three protocols adopted by TPM 2.0 are not secure).

Although tpm.KE cannot achieve the rigorous KCI resistance and PFS properties, it can satisfy weak forms of the two properties: (1) constrained KCI; that is, the control of a party's long-term key handle doesn't enable the attacker to impersonate *other, uncontrolled, parties* to the party; and (2) the constrained PFS property; that is, the expired session keys established before the attacker controls the handle of the long-term key cannot be recovered. To prove weak forms of the two properties, all is needed is to note that the proof of tpm.KE in Section 7 still holds if we allow the attacker to corrupt $\hat{A}$ and $\hat{B}$ which are the related parities of the test session, i.e., all the simulators don't abort when $\hat{A}$ and $\hat{B}$ are corrupted. The proof remains valid since the abort operations are never used in the proof.

8.2 Suggestions

TPM 2.0 is an important industrial specification which might be deployed widely in practice, so a formal analysis of its key exchange primitive is critical. In this work we formally show that tpm.KE can achieve the basic security property defined by modern AKE models. However, this goal is achieved under some constraints on the attacker, and if the host of a TPM doesn't code its software properly, tpm.KE would be vulnerable to attacks. In order to ensure proper use of tpm.KE, we give the following suggestions:

1. As only in the environment that all long-term keys are protected by the TPM can tpm.KE achieve rigorous security property, we suggest that the Certificate Authority only issues certificates for keys that are generated by TPM chips. This can be done via the Privacy CA protocol [10] or the direct anonymous attestation (DAA) protocol [6] if higher anonymity is required.
2. Note that the Full Unified Model scheme would be definitely insecure if the unhashed value Z_1 is compromised by the attacker. We suggest that the software running on the host which derives the session key from the return results of the TPM should be well protected, and the software should delete the return results of the TPM (especially Z_1 of the Full Unified Model scheme) immediately after the session key is derived.

In real world environments, it's common that some parties are equipped with the TPM and others are not, and some CAs only issue certificates for keys protected by the TPM (for example, via Privacy CA or DAA protocol) and some CAs issue certificates for keys no matter whether they are protected by the TPM. For the keys that are not protected by the TPM, it's feasible for the attacker to obtain their plaintexts, and these keys open a window of opportunity to mount Kaliski's UKS attack and Xu's attacks on tpm.KE: the attacker can register specific long-term keys or long-term keys whose plaintexts are available to him to compromise sessions of other honest parties (see details of these attacks described in Appendixes A and B). So current tpm.KE is not suitable for use in real world environments. For the sake of enabling tpm.KE to achieve rigorous security in real world environments, where plaintexts of some parties' long-term private keys are vulnerable to attacks, we give the following suggestions:

1. Perform the session key derivation in the TPM rather than on the host, i.e., perform $H_1()$ and $H_2()$ in the TPM. This modification to tpm.KE only adds a hash to the TPM which is negligible compared to the elliptic curve scalar multiplication. We have shown that protecting the unhashed value Z_1 is a basic requirement for the security of the Full Unified Model protocol. Protecting the unhashed value Z is also necessary for the security of MQV and SM2 key exchange in real world environments: it has been shown in [17] and [34] that the disclosure of Z of a session can lead to the vulnerability of other sessions. That's why Krawczyk mandates the hashing of Z in the HMQV (a proven secure variant of MQV), and Zhao et al. suggest putting the session key derivation of SM2 key exchange into the TPM.
2. Replace $avf()$ and $avf'()$ with cryptographic hash functions. Although we have shown that $avf()$ and $avf'()$ can be modeled as random oracles as they provide strong enough randomness, it's still preferred to replace them with cryptographic hash functions.

9 Conclusions and Future work

In this paper, we present a formal analysis of the key exchange primitive of TPM 2.0 in a unified way. One feature of our analysis is that we eliminate specific assumptions on the representation of group elements by measuring the entropy contained in the output of the $avf()$ and $avf'()$. The entropy measurement results enable us to model $avf()$ and $avf'()$ as random oracles convincingly. Another feature of our analysis is that we consider protections provided by the TPM. Our analysis shows that the TPM 2.0 indeed can provide a proven secure key exchange functionality if the following requirements are satisfied: all honest parties use the TPM (or other hardware security tokens) to protect their long-term keys, and the CA only issues certificates for keys from legitimate TPMs. However, these requirements are somewhat impractical, which limit the use of tpm.KE in real world environments. So we give suggestions on how to improve the security level of tpm.KE to enable its use in real world environments. A formal security analysis of the improved tpm.KE based on our unified security model can be done in the future work.

References

1. GM/T 0003.5-2012: Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves Part 5: Parameter definition.
2. ISO/IEC 15946-5:2009 Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 5: Elliptic curve generation.
3. W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):242–273, 2004.
4. E. B. Barker, D. Johnson, and M. E. Smid. NIST SP 800-56A. recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised). 2007.
5. S. Blake-Wilson and A. Menezes. Unknown key-share attacks on the station-to-station (sts) protocol. In *Public Key Cryptography*, pages 154–170. Springer, 1999.
6. E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145. ACM, 2004.
7. 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.
8. 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.
9. L. Chen. Recommendation for key derivation using pseudorandom functions. *NIST Special Publication*, 800:108, 2008.
10. L. Chen and B. Warinschi. Security of the tcg privacy-ca solution. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 609–616. IEEE, 2010.
11. T. Dierks. The transport layer security (tls) protocol version 1.2. 2008.
12. FIPS, PUB. 186-2. Digital Signature Standard (DSS). *National Institute of Standards and Technology (NIST)*, 2000.
13. 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.
14. 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.
15. 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.
16. 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.
17. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.
18. B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16. Springer, 2007.
19. K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *Public Key Cryptography – PKC 2006*, pages 378–394. Springer, 2006.
20. 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.

21. T. Matsumoto and Y. Takashima. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.
22. 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.
23. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – EUROCRYPT'96*, pages 387–398. Springer, 1996.
24. SEC, SECG. 2: Recommended elliptic curve domain parameters. *http://www.secg.org*, 2000.
25. Skipjack and NIST. KEA algorithm specifications, 1998.
26. TCG. TCG Algorithm Registry Family 2.0, Level 00 Revision 01.15, 2014.
27. TCG. Trusted platform module library part 1: Architecture, family 2.0, level 00 revision 01.07, 2014.
28. TCG. Trusted Platform Module Library Part 3: Commands Family 2.0, Level 00 Revision 01.07, 2014.
29. B. Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Designs, Codes and Cryptography*, 46(3):329–342, 2008.
30. F. M. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context-tree weighting method: Basic properties. *Information Theory, IEEE Transactions on*, 41(3):653–664, 1995.
31. J. Xu and D. Feng. Comments on the SM2 key exchange protocol. In *Cryptology and Network Security*, pages 160–171. Springer, 2011.
32. A. C. Yao and Y. Zhao. A new family of implicitly authenticated diffie-hellman protocols. Technical report.
33. A. C.-C. Yao and Y. Zhao. OAKE: a new family of implicitly authenticated diffie-hellman protocols. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1113–1128. ACM, 2013.
34. S. Zhao, L. Xi, Q. Zhang, Y. Qin, and D. Feng. Security analysis of SM2 key exchange protocol in TPM2. 0. *Security and Communication Networks*, 8(3):383–395, 2015.

A Kaliski's UKS attack

We describe Kaliski's UKS attack here to show how the attacker $\mathcal{M}$ successfully mounts the attack by cleverly computing its long-term private key c and the ephemeral public key X'.

1. $\hat{A}$ sends an ephemeral public key X to $\hat{B}$.
2. $\mathcal{M}$ intercepts X.
3. $\mathcal{M}$ registers with the CA a key $C = g^c$ where c is cleverly computed by the following steps:
   (a) Choose $u \in_R Z_q$;
   (b) Compute $d = avf(X)$, $X' = XA^d g^{-u}$, $e = avf(X')$, and $c = u/e$.
4. $\mathcal{M}$ then sends X' to $\hat{B}$ as the identity of $\mathcal{M}$.
5. $\mathcal{M}$ relays the ephemeral key Y from $\hat{B}$ to $\hat{A}$.

Note that $X'C^e = XA^d$, and therefore the keys computed in sessions $(\hat{A}, \hat{B}, X, Y)$ and $(\hat{B}, \mathcal{M}, Y, X')$ are identical.

B Xu's attacks

Here we describe Xu's two attacks on the SM2 key exchange protocol to show that in the first attack the attacker $\mathcal{M}$ requires to register a specific long-term public key, and in the second attack $\mathcal{M}$ needs to use the private key of $\hat{A}$ to perform some computations.

B.1 Attack I

$\mathcal{M}$ selects $u \in_R Z_q$, and registers this cleverly computed public key $M = Ag^u$.

1. $\hat{A}$ sends an ephemeral public key X to $\hat{B}$.
2. $\mathcal{M}$ intercepts X and sends it to $\hat{B}$ as the identity of $\mathcal{M}$.
3. $\hat{B}$ sends its ephemeral public key Y to $\mathcal{M}$, and computes $Z_B = (MX^d)^{h(b+ey)}$ where $d = avf'(X)$ and $e = avf'(Y)$.
4. $\mathcal{M}$ forwards Y to $\hat{A}$ as the identity of $\hat{B}$. $\hat{A}$ computes $Z_A = (BY^e)^{h(a+dx)}$.
5. $\mathcal{M}$ corrupts Z_B of session $(\hat{B}, \mathcal{M}, Y, X)$, then $\mathcal{M}$ can compute Z_A of session $(\hat{A}, \hat{B}, X, Y)$: $Z_A = Z_B/(BY^e)^{hu}$, and further derives the session key of $(\hat{A}, \hat{B}, X, Y)$.

Note that above attack shows that corruption of session $(\hat{B}, \mathcal{M}, Y, X)$ does affect the security of session $(\hat{A}, \hat{B}, X, Y)$, so the SM2 key exchange protocol cannot achieve the security defined by modern AKE security models.

B.2 Attack II

$\mathcal{M}$ first registers a legal key $M = g^m$.

1. $\hat{A}$ sends an ephemeral public key X to $\hat{B}$.
2. $\mathcal{M}$ intercepts X and sends $X' = AX^d$ $(d = avf'(X))$ to $\hat{B}$ as the identity of $\mathcal{M}$.
3. $\hat{B}$ sends an ephemeral public key Y to $\mathcal{M}$, and computes $Z_B = (MX'^{d'})^{h(b+ey)}$ where $d' = avf'(X')$ and $e = avf'(Y)$.
4. $\mathcal{M}$ forwards it to $\hat{A}$. $\hat{A}$ computes $Z_A = (BY^e)^{h(a+dx)}$ where $d = avf'(X)$ and $e = avf'(Y)$.
5. $\mathcal{M}$ corrupts Z_B of session $(\hat{B}, \mathcal{M}, Y, X')$ and computes Z_A of session $(\hat{A}, \hat{B}, X, Y)$: $Z_A = (Z_A/(BY^e)^{hm})^{d'^{-1}}$, and further derives the session key of $(\hat{A}, \hat{B}, X, Y)$.

C Group Representation Attack on MQV

For the benefit of the reader we present the group representation attack on MQV here. Consider such a group G that the representation of its elements satisfies that the $\lceil q/2 \rceil$ LSBs of the representation of points' x-coordinate are fixed. We use c to denote this fixed value. In this case, the Z value of MQV becomes $Z = g^{h(x+ca)(y+cb)}$. The attacker $\mathcal{M}$ can launch the following attack:

1. $\mathcal{M}$ randomly chooses $x^* \in_R Z_q$, and computes $X^* = g^{x^*}/A^c$.
2. $\mathcal{M}$ sends X^* to $\hat{B}$ as the identity of $\hat{A}$.
3. $\hat{B}$ responds with $Y = g^y$, computes $Z = (X^* A^c)^{h(y+cb)}$, and computes its session key $K = H_2(Z, \hat{A}, \hat{B})$.
4. $\mathcal{M}$ can also compute the session key $K = H_2((YB^c)^{hx^*}, \hat{A}, \hat{B})$.

Above attack shows that $\mathcal{M}$ can impersonate $\hat{A}$ without knowing the private key of $\hat{A}$ because of the special representation of the group elements.