A Comprehensive Formal Security Analysis and Revision of the Two-phase Key Exchange Primitive of TPM 2.0

Qianying Zhang, Shijun Zhao, Zhiping Shi, Yong Guan, and Guohui Wang

Abstract

The Trusted Platform Module (TPM) version 2.0, which has been demonstrated as a key element of Industry 4.0, presents a two-phase key exchange primitive for secure communications between Industry 4.0 components. The key exchange primitive of TPM 2.0 can be used to implement three widely-standardized authenticated key exchange protocols: the Full Unified Model, the Full MQV, and the SM2 key exchange protocols. However, vulnerabilities have been found in all of these protocols. Fortunately, it seems that the protections offered by TPM chips can mitigate these vulnerabilities. In this paper, we present a security model which captures TPM's protections on keys and protocols' computation environments and in which multiple protocols can be analyzed in a unified way. Based on the unified security model, we give the first formal security analysis of the key exchange primitive of TPM 2.0, and the analysis results show that, with the help of hardware protections of TPM chips, the key exchange primitive indeed satisfies the well-defined security property of our security model, but unfortunately under some impractical limiting conditions, which would prevent the application of the key exchange primitive in real-world networks. To make TPM 2.0 applicable to real-world networks, we present a revision of the key exchange primitive of TPM 2.0, which can keep secure without the limiting conditions. We give a rigorous analysis of our revision, and the results show that our revision achieves not only the basic security property of modern AKE security models but also some further security properties.

Index Terms

Network security, Cryptographic protocols, Communication networks

I. Introduction

In the Industry 4.0 era, all businesses will establish global networks, and smart devices are increasingly networked via the information and communication technology (ICT) and the Internet. Massive amounts of data used to predict,control and plan for better business and societal outcomes will be exchanged between Industry 4.0 components, and the exchanged data may concern security and privacy, such as inhabitant data of automotive vehicles and smart homes, so it is important to protect the security of the communication for Industry 4.0.

The TPM, presented by the Trusted Computing Group (TCG), has been demonstrated as a key element of Industry 4.0 [34], for it can protect the trustworthiness, identity, and communication of Industry 4.0 devices and has been widely used as the hardware root of trust for the devices. The newest TPM specifications, TPM 2.0 [35], [36], allow each platform to choose the functions needed and the level of security required. This flexibility enables TPM 2.0 to be applied to different kinds of platforms from low-end embedded devices to PCs and even servers, and allows TPM 2.0 to be implemented in different types such as security chips and protected software in a trusted execution environment (TEE) [7]. Take the automotive use case of

Manuscript received April 19, 2019. (Corresponding author: Shijun Zhao)

Q. Zhang, Z. Shi, Y. Guan, and G. Wang are with the College of Information Engineering, Capital Normal University (email: {qyzhang,shizp,guanyong,ghwang}@cnu.edu.cn).

S. Zhao is with the Institute of Software, Chinese Academy of Sciences (email: zhaosj@tca.iscas.ac.cn).

This manuscript is an extension of the conference version appearing in the Proceedings of the 8th International Conference on Trust and Trustworthy Computing (Trust'15) [46]. This manuscript presents the details of the proof of the key exchange primitive of the TPM 2.0 specifications, gives concrete suggestions on how to revise the key exchange primitive of TPM 2.0 to make it applicable in real-world networks and achieve a higher level of security, and rigorously analyzes the revised key exchange primitive.

$\hat{A}: (a, A = g^a)$		$\hat{B}: (b, B = g^b)$
$X = g^x$	$\xrightarrow{X}$	$Y = g^y$
	$\xleftarrow{Y}$	

Full UM:	$K = H_1(Z_1, Z_2, \hat{A}, \hat{B}, X, Y)$, where $Z_1 = g^{ab}$, $Z_2 = g^{xy}$
Full MQV:	$Z_A = (YB^e)^{h(x+da)}, Z_B = (XA^d)^{h(y+eb)}$
	$K = H_2(Z_A, \hat{A}, \hat{B}) = H_2(Z_B, \hat{A}, \hat{B})$, where
	$d = avf(X) \stackrel{\text{def}}{=} 2^l + (X.x \bmod 2^l), e = avf(Y) \stackrel{\text{def}}{=} 2^l + (Y.x \bmod 2^l), l = \lceil q/2 \rceil$
SM2 Key Exchange:	$Z_A = (BY^e)^{h(a+dx)}, Z_B = (AX^d)^{h(b+ey)}$
	$K = H_2(Z_A, \hat{A}, \hat{B}) = H_2(Z_B, \hat{A}, \hat{B})$, where
	$d = avf'(X) \stackrel{\text{def}}{=} 2^l + (X.x \bmod 2^l), e = avf'(Y) \stackrel{\text{def}}{=} 2^l + (Y.x \bmod 2^l), l = \lfloor q/2 \rfloor$

Fig. 1: The Full UM, Full MQV, and SM2 key exchange Protocols

Industry 4.0 for example, TCG publishes TPM specifications for components of vehicle systems [37], [39]: an Automotive-Rich TPM that has rich capabilities can be defined for Head Unit or Gateway components of vehicle systems which have powerful processing, networking and applications functionality, and an Automotive-thin TPM that has fewer capabilities can be defined for Head Unit or Gateway components having limited resources.

In this paper, we focus on the secure communication feature of TPM 2.0: its authenticated key exchange (AKE) functionality, which provides secure communication interfaces for Industry 4.0 devices. AKE is an important public key primitive in modern cryptography, which allows two parties to establish a shared secret session key via a public insecure communication while providing mutual authentication. To prevent active attacks, AKE protocols usually use digital signatures or message authentication codes (MAC) to explicitly authenticate the messages exchanged [3], [9], [12], [15]. However, the authentication mechanisms incur significant overhead in both computation and communication. To overcome the disadvantages of explicitly AKE protocols, the implicitly AKE protocols [17], [19], [21]–[25], [27], [32], [40], [42]–[44] are proposed. This kind of protocols only requires basic Diffie-Hellman exchanges while providing identity authentication by combining the ephemeral keys and long-term keys during the derivation of the session key, and achieves efficiency in both computation and communication.

The TPM 2.0 specifications support three implicitly AKE protocols: the Full Unified Model (UM), Full MQV, and SM2 key exchange protocols. All the three protocols have been widely standardized: the Full UM and Full MQV protocols are standardized by ANSI [4], [5], NIST [8], IEEE [18]; the SM2 key exchange protocol is standardized by the Chinese Government State Cryptography Administration [33] and is also standardized in some industrial standards [28]. TPM 2.0 describes implicitly AKE protocols as two-phase key exchange protocols because TPM 2.0 implements them in two phases. In the first phase, the TPM generates an ephemeral DH key and sends the ephemeral key 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 a session key.

We first introduce some notations used in this paper. 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 the 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 is an elliptic curve in this work, for 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 UM, Full MQV and SM2 key exchange protocols are described in Figure 1. $H_1()$ and $H_2()$ are cryptographic hash functions. The Full UM protocol analyzed in this paper includes the ephemeral public keys exchanged as suggested by [19]. The Full MQV protocol is a variant of the original MQV protocol [27] which does not include parties' identifiers in the session key derivation.

A. Weaknesses of the AKE protocols in TPM 2.0

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

We find that the Full UM 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 presents an unknown-key share (UKS) attack [20] on the original MQV protocol: 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 $\mathcal{M}$. This UKS attack requires $\mathcal{M}$ to register a specific key $C = g^c$ with the certificate authority (CA) and send a specific ephemeral public key X' to $\hat{B}$. c and X' are so carefully computed by $\mathcal{M}$ that the session keys of sessions $(\hat{A}, \hat{B}, X, Y)$ and $(\hat{B}, \mathcal{M}, Y, X')$ are identical. The details of the UKS attack are described in Appendix A. Although the Full MQV protocol tries to prevent the above UKS attack 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., the corruption of the session $(\hat{B}, \mathcal{M}, Y, X')$ helps $\mathcal{M}$ to compromise another session $(\hat{A}, \hat{B}, X, Y)$.

Xu et al. introduce two UKS attacks [42] 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 require $\mathcal{M}$ to reveal the unhashed shared Z_B of $\hat{B}$. 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. The details of the two attacks are described in Appendix B.

The above attacks show 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 values. Unfortunately, this is exactly how the two-phase key exchange primitive of TPM 2.0 (denoted by $\mathsf{tpm.KE}$) implements these three AKE protocols: Z_1 of the Full UM protocol, the unhashed Z values of the MQV and SM2 key exchange protocols are returned to the host, whose memory is vulnerable to attacks. So it seems that $\mathsf{tpm.KE}$ is not secure.

B. Motivations and Contributions

Fortunately, protections provided by the TPM improve the security of $\mathsf{tpm.KE}$. First, all long-term keys are generated randomly in the TPM, so attackers cannot make the TPM to generate a specific key such as the carefully computed key $C = g^c$ required by Kaliski's UKS attack or $C = Ag^e$ required by Xu's first attack. Second, the TPM only provides well-defined functions through TPM commands [36] 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 prevents attackers from using a key to perform computations at will. So it seems that the above two features can prevent Kaliski's UKS attack and Xu's attacks. However, in modern cryptography, rigorous proofs of security in modern security models [11], [22] have become a basic requirement for cryptographic protocols to be standardized and are essential tools to guarantee the soundness of cryptographic protocols. As the TPM has been widely applied on kinds of computation platforms, it is important to perform formal analysis about its security mechanisms. This leads to our first motivation.

1) *How to build a security model which can precisely model the TPM's protections on keys and protocol computation environments and based on which we can perform rigorous security analysis of* $\mathsf{tpm.KE}$*?*

Although protections provided by the TPM help the MQV and SM2 key exchange protocols to resist current UKS attacks, the $avf()$ and $avf'()$ functions used in the MQV and SM2 key exchange protocols respectively make that the two protocols cannot be proven secure. Consider such a group G that the representations of its elements satisfy that the $\lceil q/2 \rceil$ least significant bits (LSBs) of the representations of points' x-coordinate are fixed. In this case, an attacker can mount the so-called group representation attacks on the MQV and SM2 key exchange protocols, in which the attacker can impersonate $\hat{A}$ to $\hat{B}$ without knowing the private key of $\hat{A}$. The group representation attack on MQV is described in Appendix C, and a similar attack on the SM2 key exchange protocol can be found in [45]. HMQV, a variant of MQV, prevents this type of attack by replacing $avf()$ with a cryptographic hash function, which enables the protocol to be proven secure in the CK model. Paper [45] also suggests replacing the $avf'()$ of the SM2 key exchange protocol with a cryptographic hash function. However, group representation attacks are not practical, for it is difficult to find an elliptic curve whose $\lceil q/2 \rceil$ LSBs of the representations of points' x-coordinate are fixed. On the contrary, the results of the $avf()$ and $avf'()$ functions seem 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 output distributions of $avf()$ and $avf'()$ on real-world elliptic curves and check whether $avf()$ and $avf'()$ provide enough entropy to prevent group representation attacks?*

As far as we know, current modern AKE security models only consider how to formally analyze one single protocol, thus all AKE protocols proven secure in the literature are analyzed separately. However, tpm.KE is designed to support three implicitly AKE protocols through unified interfaces, so we cannot use current security models to analyze tpm.KE. Besides, the design of tpm.KE brings the following security problem that should be considered in its analysis. Suppose an honest party $\hat{A}$ tries to establish a secure channel with $\hat{B}$ through MQV and an attacker controls a long-term key of $\hat{B}$'s TPM, but the type of the key is SM2. Then the question is whether the session key of $\hat{A}$ is secure if the attacker leverages the SM2 key to complete the session. Apparently, it's desirable for tpm.KE to guarantee the security of $\hat{A}$'s session key. We denote this security property by *correspondence property*. The requirements for analyzing multi-protocols simultaneously and capturing the *correspondence property* lead 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?*

Unfortunately, even if we prove tpm.KE is secure, tpm.KE only guarantees its security under the following conditions: first, all the entities in the network must use TPM chips to run the AKE protocols and if one entity uses a less secure implementation of protocols such as a software implementation, the compromise of the software implementation will affect the security of other honest entities, such as Kaliski's and Xu's UKS attacks mentioned above; second, attackers are assumed to be unable to obtain the information inside the TPM, such as long-term keys. The conditions are due to the fact that the above security model creates all protocol instances based on the TPM and prohibits attackers from obtaining information inside the TPM even they compromise the TPM. However, the above conditions are not practical for real-world networks: first, it is unrealistic to assume that all devices are protected by the TPM, and there may exist some devices whose execution environments are not secure and can be easily compromised; second, it is also unrealistic to assume that no TPM is ever compromised because attackers can launch sophisticated attacks such as invasive attacks, semi-invasive attacks or side-channel attacks to compromise TPM chips. The above unrealistic assumptions lead to our fourth motivation:

4) *How to revise the current version of* tpm.KE *so that it can be applied in real-world networks where devices may not be protected by TPM chips and TPM chips may be compromised?*

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

1) We leverage the min-entropy, a notion in the information theory, to quantitatively evaluate the amount of randomness in the output distributions of $avf()$ and $avf'()$. We evaluate several series of elliptic curves used in practice, covering all elliptic curves adopted by the TPM 2.0 algorithm specification [38]. The evaluation 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 basic security property defined by modern AKE security 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 UM functionality of TPM 2.0 or properly implementing the host's software.
4) The tpm.KE is proven secure under some limiting conditions, resulting in some restrictions on the usage of tpm.KE, so we give suggestions on how to use tpm.KE properly to achieve a secure implementation of AKE protocols.
5) The limiting conditions required by the current version of tpm.KE are impractical for real-world networks, so we present a revision of tpm.KE to eliminate the limiting conditions and give concrete suggestions on how to revise the current version of TPM 2.0 specifications.
6) We rigorously analyze our revision of tpm.KE, and the analysis results show that our revision achieves not only the basic security property defined by modern AKE security models but also some further security properties: resistance to key-compromise impersonation (KCI) and weak Perfect Forward Secrecy (PFS).

C. Organization

In the rest of this paper, Section II gives some preliminaries. Section III introduces the two-phase key exchange primitive defined by TPM 2.0 specifications, gives a quantitative measure of several series of elliptic curves used in practice, and presents an informal analysis of tpm.KE. Section IV presents our unified security model for tpm.KE. Section V gives a formal description of tpm.KE. Section VI proves the unforgeability of the MQV and SM2 key exchange functionalities provided by tpm.KE, and it can simplify our security proof. Section VII formally analyzes the basic security of tpm.KE in our new model. Section VIII discusses the KCI-resistance and weak PFS properties of tpm.KE. Section X presents our revision of tpm.KE. Sections XI and XII rigorously analyze our revision of tpm.KE. Section XIII discusses the KCI-resistance and weak PFS properties of our revision. Section XIV concludes this paper.

II. Preliminaries

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

A. Min-entropy

Min-entropy is a notion in 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 an attacker to perform group representation attacks. Take the following two extreme cases for example: if the min-entropy of a random variable is equal to 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 two popular 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 of min-entropy for one binary bit is: $H = -log_2^{p_{max}}$ where $p_{max} = max\{p_0, p_1\}$ and p_0, p_1

are probabilities that 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) [41] is an optimal compression algorithm for stationary sources and is usually used to estimate the min-entropy.

B. 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 G and its 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 can solve the CDH problem, even if the algorithm is provided with a DDH-oracle for G.

III. The TPM Key Exchange Primitive

This section first describes tpm.KE and the related TPM commands, and then gives an informal analysis of tpm.KE. In the informal analysis, we present solutions to prevent impersonation attacks on the Full UM protocol, and a quantitative measure of the randomness of the output distributions of $avf()$ and $avf'()$ on all the elliptic curves adopted by the TPM 2.0 specifications.

A. Introduction of tpm.KE

tpm.KE consists of two phases. In the first phase, the TPM generates an ephemeral key which will be transferred to the other party by the TPM's host. In the second phase, the TPM generates the unhashed secret value according to the specification of the selected protocol, and the host derives the session key from the unhashed secret value. Before the two phases, the **Key Generation** procedure should be invoked to generate a long-term key.

- **Key Generation**. The commands TPM2_Create() and TPM2_CreatePrimary() are used to generate long-term keys. 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.
- **First Phase.** The command TPM2_EC_Ephemeral() is used to generate an ephemeral key, and it performs the following steps:
  1) Generate $x = \mathsf{KDFa}(Random, Count)$, where KDFa() is a key derivation function [13], $Random$ is a secure random value stored inside the TPM, and $Count$ is a counter.
  2) Set $ctr = Count$, $A[ctr] = 1$, and $Count = Count + 1$, 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 does not need to store the ephemeral private key x as it can be recovered using KDFa() and ctr.

- **Second Phase.** The command related to this phase is TPM2_ZGen_2Phase(), and it is the main command of tpm.KE. This command takes the following items as input:

$scheme$	A protocol scheme selector.
$keyA$	The key handle identifying the long-term private key a.
ctr	The counter used to identify the ephemeral key generated in the first phase.
B	The public key of $\hat{B}$, with whom $\hat{A}$ 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) Clear $A[ctr]$: set $A[ctr] = 0$ to ensure that the ephemeral private key x can only be used once.
   c) Return Z_1 and Z_2.
3) Finally, the host computes the session key leveraging the unhanshed values Z_1 and Z_2 returned by the TPM.

B. Informal Analysis

In Sections I-A and I-B, we show two weaknesses of tpm.KE which prevent it from achieving the basic security property defined by modern AKE security models. One weakness is that tpm.KE returns Z_1 of the Full UM protocol to the host whose memory can be compromised, which makes Z_1 be available to the attacker. The other one is the weakness caused by $avf()$ and $avf'()$, which results in group representation attacks against the MQV and SM2 key exchange protocols. We give two solutions to overcome the first weakness:

1) Perform the entire session key computation of Full UM 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 the kernel mode, and delete Z_1 and Z_2 as soon as the session key is derived.

The first solution requires modifying TPM 2.0 specifications, and the second one requires that the software deriving the session key should be implemented properly and included in the Trusted Computing Base (TCB).

As it seems that the second weakness only happens in theory, we perform a quantitative measure of the min-entropy contained in the output distributions of $avf()$ and $avf'()$ to check whether this weakness can happen in the real world. We measure several series of widely deployed elliptic curves: the NIST series [16], the BN series [2], the SECG series [31], and an SM2 elliptic curve [1]. Our measure totals 17 elliptic curves and covers all elliptic curves adopted by TPM 2.0 [38]. We generate 16384 points for each elliptic curve, apply $avf'()$ to the points of the SM2 P256 curve, and apply $avf()$ to the points of the rest curves. 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 using the methods

		NIST Series					BN Series						SECG Series					SM2
		P192	P224	P256	P384	P521	P192	P224	P256	P384	P521	P638	P192	P224	P256	P384	P521	P256
NIST 800-89	$avf()$	95.2	111.0	126.9	190.2	258.7	95.1	111.0	126.9	190.2	253.6	315.0	95.2	111.0	126.6	190.4	258.6	125.8
	SHA-2	95.9	112.0	127.9	191.3	259.8	96.2	112.0	128.0	191.2	254.8	316.1	96.0	11.0	127.9	191.3	259.6	126.9
CTW Ratio	$avf()$	97%	98%	98%	99%	100%	97%	98%	98%	99%	99%	100%	97%	98%	98%	99%	100%	100%
	SHA-2	98%	98%	99%	99%	100%	98%	98%	99%	99%	100%	100%	98%	98%	99%	99%	100%	100%

TABLE I: Min-entropy results

of NIST SP 800-90 (Formula 1) and CTW compression. The measurement results are summarized in Table I. 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 distributions 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 distributions of $avf()$ and $avf'()$ are close to the uniform distribution. Take the measurement of BN P256 for example, the min-entropy calculated by the NIST's method is 126.9, very close to the output length of $avf()$ which is $129 = \lceil 256/2 \rceil$+1, and the CTW ratio is 98.1% which is close to 100%. Our measure indicates that the outputs of $avf()$ ($avf'()$) on different elliptic curve points are almost independent, and it is infeasible to mount group representation attacks on real-world elliptic curves. So we model $avf()$ and $avf'()$ as random oracles in our formal analysis.

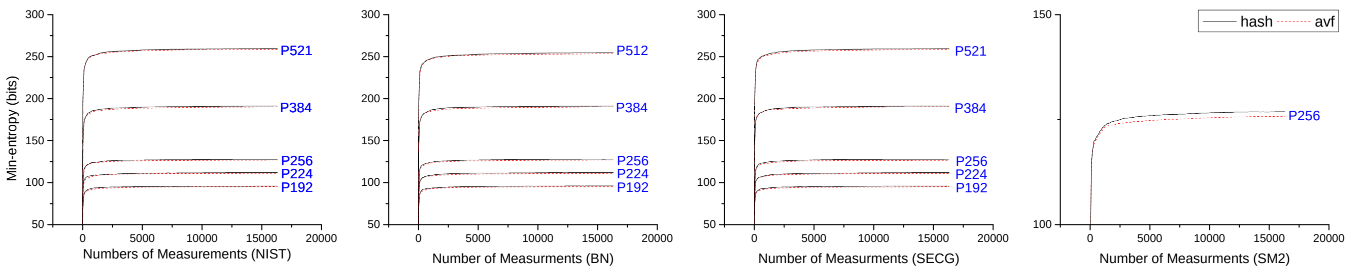


Fig. 2: Min-entropy evaluation

IV. The Unified Security Model

This section illustrates our unified security model for tpm.KE and the attacker model which captures the capabilities of attackers. In our security model, each party has a long-term key generated by the TPM and a certificate (issued by a CA) that binds the public part of the long-term key to the identity of the 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 or responder who sends an ephemeral public key to its peer party, the ephemeral key is generated 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, and Y the incoming ephemeral public key. This kind of session identifier cannot identify a session established by tpm.KE because 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 the 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 the session $(sc, \hat{A}, \hat{B}, X, Y)$.

The sc of 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 do not capture this attack because they do not support formal analysis of multiple kinds of protocols in a unified way. However, this attack can happen on tpm.KE because it supports three AKE schemes and the TPM 2.0 specifications do not force the TPM to check the key type of its peer party. We say tpm.KE satisfies *correspondence property* if it can resist the above attack, i.e., the session $(sc, \hat{A}, \hat{B}, X, Y)$ is secure if its message-matching session is compromised.

Attacker Model. The model 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)**: $\mathcal{M}$ directly queries 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 $\mathsf{tpm.KE}$ 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, $\mathcal{M}$ does not learn anything about the plaintext of the long-term private key but obtains the black-box access of the key via TPM interfaces.
- **Test(s)**: Pick $b \xleftarrow{R} 0, 1$. If b = 1, provide $\mathcal{M}$ with 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 the session as well as its matching session (if it exists) is not subject to the above three queries. A session is called "exposed" if $\mathcal{M}$ performs any one of the above three queries to this session.

Note that our model differs from previous AKE security models in that the **Corruption** query to some party does not 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 captures the protections provided by the TPM, which are described in Section I-B.

The security is defined based on an experiment 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 the above capabilities is called a **KE-attacker**. The formal security is defined as follows.

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

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 $\mathsf{tpm.KE}$: it guarantees that exposure of one session does not help the attacker to compromise the security of another session. Note that our security definition of $\mathsf{tpm.KE}$ 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.

V. Formal Description of tpm.KE

We let $\mathsf{ephem}_A()$ denote the interface of the first phase of $\mathsf{tpm.KE}$ where A is the long-term key of $\hat{A}$. Considering the black-box manner of TPM chips, we model the key exchange functionalities of $\mathsf{tpm.KE}$ (i.e., the second phase of $\mathsf{tpm.KE}$) as oracles: the Full UM, MQV, and SM2 key exchange functionalities 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. $\mathcal{O}_A^{\mathsf{MQV}}$ and $\mathcal{O}_A^{\mathsf{SM2}}$ take as input the input of TPM2_ZGen_2Phase() and return the unhashed values. We let $\mathcal{O}_A^{\mathsf{EC}}$ directly return the session key but not the unhashed values Z_1 and Z_2, and this modification simulates our solutions to the first weakness of $\mathsf{tpm.KE}$. We now formally describe $\mathsf{tpm.KE}$ through 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 is identified as (an 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; output Y as its outgoing ephemeral public key.
   b) With input $(sc, keyB, ctr_y, A, X)$ where $keyB$ is the key handle of B, invoke the corresponding oracle according to the value of sc:
      Case TPM_ALG_ECDH: Invoke $\mathcal{O}_B^{\mathsf{EC}}$ and set the session key K to be the return result of $\mathcal{O}_B^{\mathsf{EC}}$.
      Case 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 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 $(sc, \hat{A}, \hat{B}, X)$, and then performs the following steps:
   a) With input $(sc, keyA, ctr_x, B, Y)$ where $keyA$ is the key handle of A, invoke the corresponding oracle according to the value of sc:
      Case TPM_ALG_ECDH: Invoke $\mathcal{O}_A^{\mathsf{EC}}$ and set the session key K to be the return result of $\mathcal{O}_A^{\mathsf{EC}}$.
      Case 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 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 $(sc, \hat{A}, \hat{B}, X, Y)$.

VI. Unforgeability of MQV and SM2 Key Exchange Functionalities

In this section, we give the formal definitions of MQV and SM2 key exchange functionalities of tpm.KE and formally prove their unforgeability with a constraint on the attacker. The unforgeability can simplify our formal analysis of tpm.KE.

Definition 4 (The MQV Functionality of 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 $Z = (XA^d)^{h(y+eb)}$ where $d = avf(X)$ and $e = avf(Y)$. The challenger can verify the return result (Y, Z) with respect to the challenge X by checking whether $Z = (YB^e)^{h(x+da)}$.

Definition 5 (The 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 $Z = (AX^d)^{h(b+ey)}$, where $d = avf'(X)$ and $e = avf'(Y)$. The challenger can verify the return result (Y, Z) with respect to the challenge X by checking whether $Z = (BY^e)^{h(a+dx)}$.

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.

The constraint is reasonable because the TPM prevents attackers from obtaining the plaintext of keys in it. 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 it simulates $\mathcal{O}_B^{\mathsf{MQV}}$ as follows:

1) On receipt the 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 $(Y, Z' = (X'A^d)^{hs})$.

If $\mathcal{M}$ successfully forges a return result (Y, Z) on the pair (A, X) in an experiment, then $\mathcal{C}$ obtains $Z = (XA^d)^{h(y+eb)}$ where $d = avf(X)$ and $e = avf(Y)$. Note that without the knowledge of the private key y of Y, $\mathcal{C}$ is unable to compute CDH(X, B). Following the Forking Lemma [29] 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 $Z' = (XA^d)^{h(y+e'b)}$ and can compute CDH(X, B) $= (\frac{Z}{Z'})^{\frac{1}{h(e-e')}} B^{-da}$. ■

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 because it can be easily completed following the proof of theorem 1.

VII. Security Analysis of tpm.KE

In this section, we analyze the security of tpm.KE in the security model defined in Section IV. 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 the state of a session stores the results returned by the TPM and the information stored in the host. For the Full UM scheme, the session state is the session key; for the MQV and SM2 key exchange schemes, the session state is the unhashed values returned by the TPM.

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

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

Lemma 4: If two parties $\hat{A}$ and $\hat{B}$ complete matching sessions, then their session keys are the same.

Lemma 5: 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 4 follows immediately from the definition of matching sessions. That is, if $\hat{A}$ completes the session $(sc, \hat{A}, \hat{B}, X, Y)$ and $\hat{B}$ completes the matching session $(sc, \hat{B}, \hat{A}, Y, X)$, it's easy to verify that $\hat{A}$'s session key is the same as $\hat{B}$'s according to the specifications of the protocols (Figure 1).

The rest of this section proves Lemma 5. Let $\mathcal{M}$ be an 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).

A. Infeasibility of Forging Attacks

Consider a successful forging attack performed by $\mathcal{M}$. Let $(sc, \hat{A}, \hat{B}, X_0, Y_0)$ be the test session for whose tuple $\mathcal{M}$ outputs a correct guess. By the convention on session identifiers, we know that the test session is held by $\hat{A}$, 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, 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 analyze the three cases separately in the following.

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 parties $\hat{A}$ and $\hat{B}$ from the honest parties and sets their public keys to be A and B respectively, and 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 V. $\mathcal{M}$ sets the type of all long-term keys. If the type of A is not TPM_ALG_ECDH, $\mathcal{S}_1$ aborts. $\mathcal{S}_1$ creates oracles modeling the two-phase key exchange functionalities for each party normally except parties $\hat{A}$ and $\hat{B}$ because it possesses the long-term private keys 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 $sc =$ TPM_ALG_ECDH and $\hat{P}_2 = \hat{B}$. 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 $sc =$ TPM_ALG_ECDH and $\hat{P}_2 = \hat{B}$. 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 $H_1(Z_1, Z_2, \hat{A}, \hat{P}, X, Y)$ where $Z_1 = A^p$ (p is the long-term private key of $\hat{P}$) 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 TPM_ALG_ECDH: Model following the description in Section V.
   b) Case TPM_ALG_ECMQV:
      i) Randomly choose $e, s \in Z_q$.
      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 TPM_ALG_SM2:
      i) Randomly choose $e, s \in Z_q$.

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 $\mathcal{O}_B^{\mathsf{EC}}$ to model the two-phase key exchange functionality for party $\hat{B}$ according to the type of B:
   a) Case 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 TPM_ALG_ECMQV:
      i) Check whether (1) $sc =$ TPM_ALG_ECMQV, (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 the above checks succeed, continue, else return an error.
      ii) Suppose the record in T indexed by ctr_y is $(ctr_y, e, s, Y, -)$, set $Z_1 = (XP^d)^{hs}$ where $d = avf(X)$, and set the last element of the record to be $\times$.
      iii) Return $(Z_1, Z_2 = NULL)$.
   c) Case TPM_ALG_SM2:
      i) Check whether (1) $sc =$ TPM_ALG_SM2, (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 the above checks pass, continue, else return an error.
      ii) Suppose the record in T indexed by ctr_y is $(ctr_y, e, s, Y, -)$, set $Z_1 = (PX^d)^{hs}$ where $d = avf'(X)$, and set the last element of the record to be $\times$.
      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 succeeds by outputting CDH$(A, B) = Z_1$.
   b) If the value of the function on input σ has been defined previously, return it.
   c) If the value of $H_{spec}()$ on input $(\hat{P}_1, \hat{P}_2, X, Y)$ has been defined previously, 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 $\mathcal{S}_1$ does not 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 $\mathcal{S}_1$ does not abort in Step 1, 2, 9, 10, 11. Therefore, $\mathcal{S}_1$ simulates $\mathcal{M}$'s environment perfectly. Thus, if $\mathcal{M}$ wins with non-negligible probability in this case, the success probability of $\mathcal{S}_1$ is bounded by

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

■

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 did not 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 four 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 V. $\mathcal{M}$ 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 keys 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 $\hat{P}_2 = \hat{B}$ (or $\hat{P}_2 = \hat{A}$). 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, $\hat{P}_2 = \hat{B}$ (or $\hat{P}_2 = \hat{A}$), 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 $DDH(X_0A^d, Y_0B^e, Z^{1/h}) = 1$ where $d = avf(X_0)$ and $e = avf(Y_0)$, then $\mathcal{S}_2$ aborts $\mathcal{M}$ and succeeds by outputting CDH$(X_0, Y_0) = \frac{Z^{1/h}}{X_0^{eb} Y_0^{da} g^{deab}}$.
   b) If the value of the function on input σ has been defined previously, 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 $\mathcal{S}_2$ does not 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 $\mathcal{S}_2$ does not abort in Step 1, 2, 4, 5, 6. Therefore, $\mathcal{S}_2$ simulates $\mathcal{M}$'s environment perfectly. 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, and 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 parties $\hat{A}$ 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 $\mathsf{ephem}_P()$ for all parties following the description in Section V. $\mathcal{M}$ sets the types 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 keys 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 $\hat{P}_2 = \hat{B}$. 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 $\mathsf{ephem}_B()$ 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.
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, $\hat{P}_2 = \hat{B}$, 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, $\hat{P}_2 = \hat{A}$, 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 $\mathcal{S}_3$ does not 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 $\mathcal{S}_3$ does not abort in Step 1, 2, 4, 5, 6. Therefore, $\mathcal{S}_2$ simulates $\mathcal{M}$'s environment perfectly except with negligible probability. By the assumption, $\mathcal{M}$ correctly guesses the tuple $(Z = (Y_0 B^e)^{h(x_0+da)}, \hat{A}, \hat{B})$ 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 $Z = (Y_0 B^e)^{h(x_0+da)} = (X_0 A^d)^{h(y_0+eb)}$.
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 $\hat{B}$, $\mathcal{M}$ does not know a and b. So $\mathcal{M}$ does not 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, and 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 parties $\hat{A}$ 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 V. $\mathcal{M}$ sets the type for all long-term keys. If the type of A is not TPM_ALG_ECMQV, $\mathcal{S}_4$ 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 keys 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 $\hat{P}_2 = \hat{B}$. 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 $\hat{P}_2 = \hat{B}$. 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 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 TPM_ALG_ECMQV, so $\mathcal{S}_4$ does not 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 $\mathcal{S}_4$ does not abort in Step 1, 2, 4, 5, 6. Therefore, $\mathcal{S}_4$ simulates $\mathcal{M}$'s environment perfectly. By the assumption, $\mathcal{M}$ correctly guesses the tuple $(Z = (Y_0 B^e)^{h(x_0+da)}, \hat{A}, \hat{B})$ 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 $Z = (Y_0 B^e)^{h(x_0+da)} = (X_0 A^d)^{h(y_0+eb)}$.
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 TPM_ALG_ECDH or TPM_ALG_SM2, $\mathcal{O}_B^{\mathsf{MQV}}$ has never been created by $\mathcal{S}_4$.
   b) If the type of B is TPM_ALG_ECMQV, $\mathcal{S}_4$ 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), $\hat{B}$ must have a session identified by $(sc =$ 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 $\hat{B}$, $\mathcal{M}$ does not know a and b. So $\mathcal{M}$ does not 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 $\mathcal{S}_4) \geq \frac{1}{3n^2k} Pr(\mathcal{M})$. ■

3) Analysis of Case 3: The analysis of Case 3 is similar to Case 2. It is easy to get a full proof by following the analysis in Section VII-A2, so we omit the analysis of Case 3.

B. Infeasibility of Key-replication Attacks

If $\mathcal{M}$ successfully launches a key-replication attack against the test session $s = (sc, \hat{A}, \hat{B}, X_0, Y_0)$, he 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 s. The sc of s' must fall under one of the following three cases. By demonstrating that key-replication attacks are impossible in any of the three cases, we prove that $\mathcal{M}$ cannot launch key-replication attacks.

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

1) Analysis of Case 1: 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 TPM_ALG_ECDH. So 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 $sc =$ 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.

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 unforgeability of MQV functionality or SM2 key exchange functionality.

Since s' has the same session key as the test session s, s' must have the same σ as the test session. In all the subcases of Case 2 in Section VII-A2, all the simulators ($\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$) provide $\mathcal{M}$ with the session state of all exposed sessions. Therefore, $\mathcal{M}$ can obtain the 3-tuple 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 the 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 directly get the analysis of Case 3.

VIII. Further Security Properties of tpm.KE

Besides the basic security property defined by modern security models, it is desirable for AKE protocols to achieve the following two security properties. The key-compromise impersonation (KCI) resistance property: the knowledge of a party's long-term private key does not enable the attacker to impersonate *other uncorrupted parties* to the party. The Perfect Forward Secrecy (PFS) property: the expired session keys established before the compromise of the long-term key cannot be recovered.

Our security model does not capture the KCI-resistance property and the PFS property, for our attacker model does not allow the attacker to obtain the plaintext of the long-term private key but only allows the attacker to control the handle of it. Although tpm.KE cannot achieve the rigorous KCI resistance and PFS properties, it can satisfy weak forms of the two properties: (1) *constrained KCI-resistance*; that is, the control of a party's long-term key handle does not 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 the above weak forms of the two properties, all needed is to note that the proof of tpm.KE in Section VII still holds if we allow the attacker to corrupt $\hat{A}$ and $\hat{B}$ which are the related parties of the test session, i.e., all the simulators do not abort when $\hat{A}$ and $\hat{B}$ are corrupted. The proof remains valid since the abort operations are never used in the proof.

IX. Suggestions on Usage of the Current Version of tpm.KE

Although we formally prove that tpm.KE satisfies the basic security property defined by modern AKE models, this is achieved under the following conditions: first, all the long-term keys must be generated by the TPM, and the attacker cannot register arbitrary keys; second, the attacker cannot obtain the plaintext of the long-term key even he corrupts the party; third, the attacker cannot obtain the unhashed value of the Full UM protocol. These constraints require that engineers should deploy tpm.KE properly, otherwise tpm.KE would be unable to provide secure communications. In order to help engineers to use tpm.KE securely, we give the following suggestions:

1) For a network that plans to use tpm.KE to protect its communications, we suggest that all devices in the network should be equipped with the TPM and the CA should only issue certificates for keys that are generated by the TPM. This requires the CA to check the validity of the TPM, and this

can be done by leveraging the Privacy CA protocol [14] or the direct anonymous attestation (DAA) protocol [10] if higher anonymity is required.

2) The network administrator should guarantee that all TPM chips are well protected against sophisticated physical attacks which can obtain the secrets inside the TPM, and the administrator should know that if one TPM chip is physically broken, the whole network may no longer be secure.
3) The software running on the host which derives the session key from the return results of the TPM should be well protected. For example, run the software in the secure execution environment provided by the Intel SGX [6], [26] or ARM TrustZone [7] technologies and delete the return results of the TPM (especially Z_1 of the Full UM protocol) immediately after the session key is derived.

X. Revision of tpm.KE

A real-world network may contain various kinds of devices, and it is common that some devices are protected by the TPM and the others are not. Moreover, there exist physical attacks that can access all keys inside the TPM [30]. So it is hard for real-world networks to satisfy the conditions required to ensure tpm.KE's security. Therefore, the current version of tpm.KE puts a significant limitation on its application on real-world networks.

In order to make tpm.KE more applicable to real-world networks, we suggest revising TPM 2.0 specifications as follows: *perform the session key derivation in the TPM rather than on the host, i.e., perform* $H_1()$ *and* $H_2()$ *in the TPM*. The revision only adds a hash computation to the TPM, which is negligible compared to the elliptic curve scalar multiplication. This revision is inspired by the HMQV protocol, the first proven secure implicitly-AKE protocol, which requires that its unhashed value Z should be stored in the same secure environment as the long-term key and only the session key is output outside of the environment. We give concrete suggestions on how to revise TPM 2.0 specifications: the only change is to revise TPM2_ZGen_2Phase() command in the TPM 2.0 Command specification [36]. The command is revised to derive the session key inside the TPM and return the session key, while the original command returns the unhashed values Z_1 and Z_2. Figure 3 describes the changes to the input and response of the TPM2_ZGen_2Phase() command.

- **Changes of input**: add the identities of the owner and peer of the session.
- **Changes of response**: not return the unhashed values but the session key.

Type	Name
COMMAND_TAG	tag
UINT32	cmdSize
TPM_CC	cmdCode
TPMI_DH_OBJECT	keyA
TPM2B_ECC_POINT	B
TPM2B_ECC_POINT	Y
KEY_EXCHANGE	scheme
UINT16	ctr
TPM2B_MAX_BUFFER	identityA
TPM2B_MAX_BUFFER	identityB

(a). TPM2_Zgen_2Phase() Input

Type	Name
TPM_ST	tag
UINT32	responseSize
TPM_RC	responseCode
TPM2B_ECC_POINT	outZ1
TPM2B_ECC_POINT	outZ2

Type	Name
TPM_ST	tag
UINT32	responseSize
TPM_RC	responseCode
TPM2B_SYM_KEY	sessionKey

(b). TPM2_Zgen_2Phase() Response

Fig. 3: Revision of TPM2_ZGen_2Phase()

XI. Preparation for the Security Analysis of the Revision

In this section, we do the preparation work for the security analysis of the revision of tpm.KE (denoted by tpm.KE.rev), including the formal description of tpm.KE.rev, the session state, and the attacker model.

A. *Formal Description of* tpm.KE.rev

The first phase of tpm.KE.rev is the same as tpm.KE, so we use the same $\mathsf{ephem}()_A$ to model the interface of the first phase of tpm.KE.rev. As the TPM returns the session key in the second phase of $\mathsf{tpm.KE.rev}$, we modify the oracles $\mathcal{O}_A^{\mathsf{MQV}}$ and $\mathcal{O}_A^{\mathsf{SM2}}$ to return the session key. We get the formal description of $\mathsf{tpm.KE.rev}$ by using the modified oracles $\mathcal{O}_A^{\mathsf{MQV}}$ and $\mathcal{O}_A^{\mathsf{SM2}}$ to replace the original oracles of the formal description of $\mathsf{tpm.KE}$ in Section V.

As our formal analysis of $\mathsf{tpm.KE.rev}$ contains parties that are not equipped with the TPM, we need to describe how the protocols are implemented in these parties. These parties also use the *Initiate*, *Respond* and *Complete* activations to construct protocol sessions. The three activations for these parties differ frosm the activations for $\mathsf{tpm.KE.rev}$ in that they do not leverage the interfaces of the TPM, i.e., $\mathsf{ephem}()_A$, $\mathcal{O}_A^{\mathsf{EC}}$, $\mathcal{O}_A^{\mathsf{MQV}}$, and $\mathcal{O}_A^{\mathsf{SM2}}$, but run following the specifications of the protocols.

B. Session State

The session state includes the information returned by the TPM. As the ephemeral key and unhashed values (such as Z_1 and Z_2) are stored inside the TPM, we specify that the session state only includes the session key.

C. Attacker Model for tpm.KE.rev

The attacker model for $\mathsf{tpm.KE.rev}$ is used to capture realistic attack capabilities in real-world networks, which are much stronger than the attack capabilities captured by the model for $\mathsf{tpm.KE}$. In particular, the model for $\mathsf{tpm.KE.rev}$ allows the attacker to register arbitrary keys to the CA and obtain the plaintext of a long-term key by corrupting the party. To capture the above stronger attack capabilities, we add the *EstablishParty* query to the model for $\mathsf{tpm.KE}$ (Section IV) and modify the *SessionStateReveal* and *Corruption* queries.

- **EstablishParty($\hat{P}$)**: this query is newly added to the model to allow the attacker to register a static public key on behalf of party $\hat{P}$.
- **SessionStateReveal(s)**: $\mathcal{M}$ directly queries at session s while still incomplete and learns the session state for s. The session state only includes the session key, while the session state in the model for $\mathsf{tpm.KE}$ includes the unhashed values.
- **Corruption($\hat{P}$)**: this query allows $\mathcal{M}$ to learn the plaintext of the long-term private key of party $\hat{P}$, while this query in the model for $\mathsf{tpm.KE}$ only allows the attacker to obtain the black-box access of the long-term key.

XII. Security Analysis of tpm.KE.rev

In order to make our analysis be close to real-world networks as much as possible, we do not make any assumption whether the parties in the network are equipped with the TPM, including the owner and the peer of the test session.

Theorem 6: Under the GDH assumption, with hash functions $H_1()$, $H_2()$, $avf()$, and $avf'()$ modeled as random oracles, $\mathsf{tpm.KE.rev}$ is secure in the unified model.

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

Lemma 7: If two parties $\hat{A}$ and $\hat{B}$ complete matching sessions, their session keys are the same.

Lemma 8: Under the GDH assumption, there is no feasible attacker that succeeds in distinguishing the session key of an unexposed session with non-negligible probability.

The proof of Lemma 7 is the same as Lemma 4, and we focus on the proof of Lemma 8. Let $\mathcal{M}$ be any attacker against $\mathsf{tpm.KE.rev}$. $\mathcal{M}$ can launch the forging attack or key-replication attack to distinguish the session key of the test session from a random value. We will show that if either of the attacks succeeds with non-negligible probability, there exists a solver succeeding in solving the GDH problem with non-negligible probability.

A. Infeasibility of Forging Attacks

Let $(sc, \hat{A}, \hat{B}, X_0, Y_0)$ be the test session for whose tuple $\mathcal{M}$ outputs a correct guess. 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 $\mathcal{M}$ succeeds with non-negligible probability in the forging attack, 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 analyze the three cases separately in the following.

1) Analysis of Case 1: For this case, we build a GDH solver $\mathcal{S}_5$. It 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. $\mathcal{S}_5$ randomly decides whether each party is protected by the TPM or not, randomly selects two parties $\hat{A}$ 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, $\mathcal{S}_5$ randomly selects an integer $i \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows.

0) $\mathcal{S}_5$ models $\mathsf{ephem}_P()$ following the description in Section V for all parties protected by the TPM. $\mathcal{M}$ sets the types of all long-term keys. If the type of A is not TPM_ALG_ECDH, $\mathcal{S}_5$ aborts. $\mathcal{S}_5$ creates oracles modeling the two-phase key exchange functionalities for parties protected by the TPM normally except parties $\hat{A}$ and $\hat{B}$. The procedures of $H_1()$ and $H_2()$ are described below. $avf()$ and $avf'()$ are simulated as random oracles in the usual way.
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 $\hat{A}$, $\mathcal{S}_5$ checks whether $sc =$ TPM_ALG_ECDH and $\hat{P}_2 = \hat{B}$. If not, $\mathcal{S}_5$ 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 $\hat{A}$, $\mathcal{S}_5$ checks whether $sc =$ TPM_ALG_ECDH and $\hat{P}_2 = \hat{B}$. If not, $\mathcal{S}_5$ 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 $\hat{A}$, $\mathcal{S}_5$ completes the session without computing a session key.
4) Establish$(\hat{P}, P)$: $\mathcal{S}_5$ creates the party $\hat{P}$ whose public key is P and marks $\hat{P}$ as corrupted.
5) If $\hat{A}$ is protected by the TPM, with the input $(sc, keyA, ctr_x, P, Y)$, $\mathcal{S}_5$ creates the oracle $\mathcal{O}_A^{\mathsf{EC}}$ as follows: compute the session key $k = H_{spec}(\hat{A}, \hat{P}, X, Y)$. $H_{spec}()$ is simulated as a random oracle.
6) Respond$(sc, \hat{A}, \hat{P}, Y)$:
   a) If $\hat{A}$ is protected by the TPM, $\mathcal{S}_5$ simulates this session activation with the help of $\mathsf{ephem}_A()$ and $\mathcal{O}_{\hat{A}}^{\mathsf{EC}}$.
   b) If $\hat{A}$ is not protected by the TPM, $\mathcal{S}_5$ generates an ephemeral key pair $(x, X = g^x)$, outputs X, and computes the session key $k = H_{spec}(\hat{A}, \hat{B}, X, Y)$.
7) Complete$(sc, \hat{A}, \hat{P}, X, Y)$:
   a) If $\hat{A}$ is protected by the TPM, $\mathcal{S}_5$ simulates this session activation with the help of $\mathsf{ephem}_A()$ and $\mathcal{O}_{\hat{A}}^{\mathsf{EC}}$.
   b) If $\hat{A}$ is not protected by the TPM, $\mathcal{S}_5$ computes the session key $k = H_{spec}(\hat{A}, \hat{P}, X, Y)$.
8) If $\hat{B}$ is protected by the TPM, with the input $(sc, keyB, ctr_y, P, X)$, $\mathcal{S}_5$ creates the oracle modeling the two-phase key exchange functionality for party $\hat{B}$: recover the ephemeral key pair $(y, Y = g^y)$ by ctr_y, and compute the session key $k = H_{spec}(\hat{B}, \hat{P}, Y, X)$.
9) Respond$(sc, \hat{B}, \hat{P}, X)$:
   a) If $\hat{B}$ is protected by the TPM, $\mathcal{S}_5$ simulates this session activation with the help of $\mathsf{ephem}_B()$ and the oracle for $\hat{B}$ (Step 8).

b) If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_5$ simulates this session activation as follows: generate an ephemeral key pair $(y, Y = g^y)$, output Y, and compute the session key $k = H_{spec}(\hat{B}, \hat{P}, Y, X)$.

10) Complete$(sc, \hat{B}, \hat{P}, Y, X)$:

a) If $\hat{B}$ is protected by the TPM, $\mathcal{S}_5$ simulates this session activation with the help of $\mathsf{ephem}_B()$ and the oracle for $\hat{B}$ (Step 8).

b) If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_5$ sets the session key to be $k = H_{spec}(\hat{B}, \hat{P}, Y, X)$.

11) SessionStateReveal(s): $\mathcal{S}_5$ returns to $\mathcal{M}$ the session state of session s. However, if s is the i-th session at $\hat{A}$, $\mathcal{S}_5$ aborts.

12) SessionKeyReveal(s): $\mathcal{S}_5$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$, $\mathcal{S}_5$ aborts.

13) Corruption$(\hat{P})$: $\mathcal{S}_5$ gives $\mathcal{M}$ the long-term key pair of $\hat{P}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}_5$ aborts.

14) $H_1(\sigma)$ 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 $DDH(A, B, Z_1) = 1$, $\mathcal{S}_5$ aborts $\mathcal{M}$ and succeeds by outputting CDH$(A, B) = Z_1$.

b) If the value of the function on input σ has been defined previously, return it.

c) If the value of $H_{spec}()$ on input $(\hat{P}_1, \hat{P}_2, X, Y)$ has been defined previously, return it.

d) Pick a key k randomly from the key distribution, and define $H_1(\sigma) = k$.

15) $H_2(\sigma)$ function for some $\sigma = (Z, \hat{P}_1, \hat{P}_2)$ proceeds as follows:

a) If the value of the function on input σ has been defined previously, return it.

b) If not defined, go over all the previous calls to $H_{spec}()$ and for each previous call of the form $H_{spec}(\hat{P}_1, \hat{P}_2, X, Y) = v$ proceed as follows according to the type of P_1.

i) Case TPM_ALG_ECMQV: check if $DDH(XP_1^d, YP_2^e, Z^{1/h}) = 1$ where $d = avf(X)$ and $e = avf(Y)$; if the condition holds, return v.

ii) Case TPM_ALG_SM2: check if $DDH(P_1X^d, P_2Y^e, Z^{1/h}) = 1$ where $d = avf'(X)$ and $e = avf'(Y)$; if the condition holds, return v.

c) If no previous calls of that form are found, pick a random key w and define $H_2(Z, \hat{P}_1, \hat{P}_2) = w$.

16) $H_{spec}(\hat{P}_1, \hat{P}_2, X, Y)$ function proceeds as follows:

a) If the value of the function on that input has been defined previously, return it.

b) If not defined, proceed as follows according to the type of P_1.

i) Case TPM_ALG_ECDH: go over all the previous calls to $H_1()$ and for each previous call of the form $H_1(Z_1, Z_2, \hat{P}_1, \hat{P}_2, X, Y) = v$ check if $DDH(P_1, P_2, Z_1) = 1$; if the condition holds, return v.

ii) Case TPM_ALG_ECMQV: go over all the previous calls to $H_2()$ and for each previous call of the form $H_2(Z, \hat{P}_1, \hat{P}_2) = v$ check if $DDH(XP_1^d, YP_2^e, Z^{1/h}) = 1$ where $d = avf(X)$ and $e = avf(Y)$; if the condition holds, return v.

iii) Case TPM_ALG_SM2: go over all the previous calls to $H_2()$ and for each previous call of the form $H_2(Z, \hat{P}_1, \hat{P}_2) = v$ check if $DDH(P_1X^d, P_2Y^e, Z^{1/h}) = 1$ where $d = avf'(X)$ and $e = avf'(Y)$; if the condition holds, return v.

c) If not found, pick a random key w, define $H_{spec}(\hat{P}_1, \hat{P}_2, X, Y) = w$.

Proof: The probability that $\mathcal{M}$ sets the type of A to be TPM_ALG_ECDH and selects the i-th session at $\hat{A}$ 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 $\mathcal{S}_5$ does not 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 at $\hat{A}$, so $\mathcal{S}_5$ does not abort in Step 1, 2, 11, 12, 13. Therefore, $\mathcal{S}_5$ simulates $\mathcal{M}$'s environment perfectly. Thus, if $\mathcal{M}$ wins with non-negligible probability in this case, the success probability of $\mathcal{S}_5$ is bounded by

$$Pr(\mathcal{S}_5) \geq \frac{1}{3n^2k} Pr(\mathcal{M}).$$

■

2) Analysis of Case 2: We build a GDH solver $\mathcal{S}_6$ for this case. $\mathcal{S}_6$ takes as input a pair (X_0, B), creates an experiment which includes n honest parties and the attacker $\mathcal{M}$, and is given access to a DDH oracle DDH. $\mathcal{S}_6$ randomly decides whether each party is protected by the TPM or not, randomly selects a party and sets its public key to be B. All the other parties compute their keys normally. Furthermore, $\mathcal{S}_6$ randomly selects an integer $i \in [1, ..., k]$. The simulation for $\mathcal{M}$'s environment proceeds as follows.

0) $\mathcal{S}_6$ models $\mathsf{ephem}_P()$ following the description in Section V for all parties protected by the TPM. $\mathcal{M}$ sets the types of all long-term keys. If the type of B is not TPM_ALG_ECMQV, $\mathcal{S}_6$ aborts. $\mathcal{S}_6$ creates oracles modeling the two-phase key exchange functionalities for parties protected by the TPM normally except $\hat{B}$. The procedures of $H_1()$ and $H_2()$ are described below. $avf()$ and $avf'()$ are simulated as random oracles in the usual way.
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 $\hat{A}$, $\mathcal{S}_6$ checks whether $sc =$ TPM_ALG_ECMQV and $\hat{P}_2 = \hat{B}$. If so, $\mathcal{S}_6$ sets the ephemeral public key to be X_0, else $\mathcal{S}_6$ aborts.
2) Respond$(sc, \hat{P}_1, \hat{P}_2, Y)$: With the exception of $\hat{B}$ (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 $\hat{A}$, $\mathcal{S}_6$ checks whether $sc =$ TPM_ALG_ECMQV and $\hat{P}_2 = \hat{B}$. If so, $\mathcal{S}_6$ sets the ephemeral public key to be X_0 and does not compute the session key, else $\mathcal{S}_6$ aborts.
3) Complete$(sc, \hat{P}_1, \hat{P}_2, X, Y)$: With the exception of $\hat{B}$, $\hat{P}_1$ executes the Complete() activation of the protocol. However, if the session is the i-th session at $\hat{A}$, $\mathcal{S}_6$ completes the session without computing a session key.
4) Establish$(\hat{P}, P)$: $\mathcal{S}_6$ creates the party $\hat{P}$ whose public key is P, and marks $\hat{P}$ as corrupted.
5) If $\hat{B}$ is protected by the TPM, with the input $(sc, keyB, ctr_y, P, X)$, $\mathcal{S}_6$ creates the oracle modeling the two-phase key exchange functionality for party $\hat{B}$: recover the ephemeral key pair $(y, Y = g^y)$ by ctr_y, and compute the session key $k = H_{spec}(\hat{B}, \hat{P}, Y, X)$.
6) Respond$(sc, \hat{B}, \hat{P}, X)$:
   a) If $\hat{B}$ is protected by the TPM, $\mathcal{S}_6$ simulates this session activation with the help of $\mathsf{ephem}_B()$ and the oracle for $\hat{B}$ (Step 5).
   b) If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_6$ simulates this session activation as follows: generate an ephemeral key pair $(y, Y = g^y)$, output Y, and compute the session key $k = H_{spec}(\hat{B}, \hat{P}, Y, X)$.
7) Complete$(sc, \hat{B}, \hat{P}, Y, X)$:
   a) If $\hat{B}$ is protected by the TPM, $\mathcal{S}_6$ simulates this session activation with the help of $\mathsf{ephem}_B()$ and the oracle for $\hat{B}$ (Step 5).
   b) If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_6$ sets the session key to be $k = H_{spec}(\hat{B}, \hat{P}, Y, X)$.
8) SessionStateReveal(s): $\mathcal{S}_6$ returns to $\mathcal{M}$ the session state of session s. However, if s is the i-th session at $\hat{A}$ or the matching session of it (if such a session exists), $\mathcal{S}_6$ aborts.
9) SessionKeyReveal(s): $\mathcal{S}_6$ returns to $\mathcal{M}$ the session key of s. If s is the i-th session at $\hat{A}$ or the matching session of it (if such a session exists), $\mathcal{S}_6$ aborts.
10) Corruption$(\hat{P})$: $\mathcal{S}_6$ gives $\mathcal{M}$ the long-term key pair of $\hat{P}$. If $\mathcal{M}$ tries to corrupt $\hat{A}$ or $\hat{B}$, $\mathcal{S}_6$ aborts.
11) $H_1(\sigma)$ function for some $\sigma = (Z_1, Z_2, \hat{P}_1, \hat{P}_2, X, Y)$ proceeds as follows:
    a) If the value of the function on input σ has been defined previously, return it.
    b) If the value of $H_{spec}()$ on input $(\hat{P}_1, \hat{P}_2, X, Y)$ has been defined previously, return it.
    c) Pick a key k randomly from the key distribution, and define $H_1(\sigma) = k$.
12) $H_2(\sigma)$ function for some $\sigma = (Z, \hat{P}_1, \hat{P}_2)$ proceeds as follows:
    a) If the value of the function on input σ has been defined previously, return it.
    b) If not defined, go over all the previous calls to $H_{spec}()$ and for each previous call of the form $H_{spec}(\hat{P}_1, \hat{P}_2, X, Y) = v$ proceed as follows according to the type of P_1.
       i) Case TPM_ALG_ECMQV: check if $DDH(XP_1^d, YP_2^e, Z^{1/h}) = 1$ where $d = avf(X)$ and $e = avf(Y)$; if the condition holds, return v.

ii) Case TPM_ALG_SM2: check if $DDH(P_1X^d, P_2Y^e, Z^{1/h}) = 1$ where $d = avf'(X)$ and $e = avf'(Y)$; if the condition holds, return v.

c) If not found, pick a random key w and define $H_2(Z, \hat{P}_1, \hat{P}_2) = w$.

13) $H_{spec}(\hat{P}_1, \hat{P}_2, X, Y)$ function proceeds as follows:

a) If the value of the function on that input has been defined previously, return it.

b) If not defined, proceed as follows according to the type of P_1.

i) Case TPM_ALG_ECDH: go over all the previous calls to $H_1()$ and for each previous call of the form $H_1(Z_1, Z_2, \hat{P}_1, \hat{P}_2, X, Y) = v$ check if $DDH(P_1, P_2, Z_1) = 1$; if the condition holds, return v.

ii) Case TPM_ALG_ECMQV: go over all the previous calls to $H_2()$ and for each previous call of the form $H_2(Z, \hat{P}_1, \hat{P}_2) = v$ check if $DDH(XP_1^d, YP_2^e, Z^{1/h}) = 1$ where $d = avf(X)$ and $e = avf(Y)$; if the condition holds, return v.

iii) Case TPM_ALG_SM2: go over all the previous calls to $H_2()$ and for each previous call of the form $H_2(Z, \hat{P}_1, \hat{P}_2) = v$ check if $DDH(P_1X^d, P_2Y^e, Z^{1/h}) = 1$ where $d = avf'(X)$ and $e = avf'(Y)$; if the condition holds, return v.

c) If not found, pick a random key w and define $H_{spec}(\hat{P}_1, \hat{P}_2, X, Y) = w$.

Proof: The probability that $\mathcal{M}$ sets the type of A to be TPM_ALG_ECMQV and selects the i-th session at $\hat{A}$ 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 TPM_ALG_ECMQV, so $\mathcal{S}_6$ does not 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 at $\hat{A}$ or its matching session (if such a session exists), so $\mathcal{S}_6$ does not abort in Step 1, 2, 8, 9, 10. Therefore, $\mathcal{S}_6$ simulates $\mathcal{M}$'s environment perfectly. If $\mathcal{M}$ wins the forging attack, he computes the 3-tuple $(Z, \hat{A}, \hat{B})$. Without the knowledge of the private key y of Y, $\mathcal{S}_6$ is unable to compute $CDH(X_0, B)$. $\mathcal{S}_6$ computes $T = \frac{Z^{1/h}}{B^a Y_0^{ae}}$ where $d = avf(X_0)$ and $e = avf(Y_0)$.

Following the Forking Lemma [29] approach, $\mathcal{S}_6$ runs $\mathcal{M}$ on the same input and same coin flips but with carefully modified answers to the $avf()$ queries. Note that $\mathcal{M}$ must have queried $avf(Y_0)$ in its first run because otherwise $\mathcal{M}$ would be unable to compute Z of the test session. For the second run of $\mathcal{M}$, $\mathcal{S}_6$ responds to $avf(Y_0)$ with a value $e' \neq e$ selected uniformly at random. If $\mathcal{M}$ succeeds in the second run, $\mathcal{S}_6$ computes $T' = \frac{Z^{1/h}}{B^a Y_0^{ae'}}$. And thereafter $\mathcal{S}_6$ computes $CDH(X_0, B) = (T'/T)^{e-e'}$. Thus, if $\mathcal{M}$ wins with non-negligible probability in this case, the success probability of $\mathcal{S}_6$ is bounded by:

$$Pr(\mathcal{S}_6) \geq \frac{C}{3n^2k} Pr(\mathcal{M})$$

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

3) Analysis of Case 3: The analysis of Case 3 is similar to Case 2. It is easy to get a full proof by following the analysis in Section XII-A2, so we omit the analysis of Case 3.

B. Infeasibility of Key-replication Attacks

If $\mathcal{M}$ successfully launches a key-replication attack against the test session $s = (sc, \hat{A}, \hat{B}, X_0, Y_0)$, he 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 s. The sc of s' must fall under one of the following three cases. By demonstrating that key-replication attacks are impossible in any of the three cases, we prove that $\mathcal{M}$ cannot launch key-replication attacks.

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

1) Analysis of Case 1: The analysis of this case is the same as the analysis in Section VII-B1 for tpm.KE.

2) Analysis of Cases 2: We prove that key-replication attacks are impossible in this case by showing that a successful key-replication attack would allow the attacker to launch forging attacks, contradicting the GDH assumption. We build a GDH solver $\mathcal{S}_7$ for this case. $\mathcal{S}_7$ takes as input a pair (X_0, B), simulates a similar environment for $\mathcal{M}$ as $\mathcal{S}_6$ does (Section XII-A2) except the simulation of session activations at $\hat{B}$ and the *SessionStateReveal* query.

- If $\hat{B}$ is protected by the TPM, $\mathcal{S}_7$ models $\mathsf{ephem}_B()$ and $\mathcal{O}_B^{\mathsf{MQV}}$ as follows:
  - $\mathcal{S}_7$ creates a Table T and models $\mathsf{ephem}_B()$ as follows.
    1) Randomly choose $e, s \in Z_q$.
    2) Set $Y = g^s/B^e$ and $e = avf(Y)$.
    3) Randomly choose an index ctr, and add a record $(ctr, e, s, Y, -)$ to T.
    4) Return ctr and Y.
  - With the input $(sc, keyB, ctr_y, P, X)$, $\mathcal{S}_7$ models $\mathcal{O}_B^{\mathsf{MQV}}$ as follows:
    1) Check whether (1) $sc =$ TPM_ALG_ECMQV, (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 the above checks succeed, continue, else return an error.
    2) Suppose the entry in T indexed by ctr_y is $(ctr_y, e, s, Y, -)$, set $Z_1 = (XP^d)^{hs}$ where $d = avf(X)$, and set the last element of the entry to be $\times$.
    3) Compute the session key $k = H_2(Z_1, \hat{B}, \hat{P})$, and return k.
- Initiate$(sc, \hat{B}, \hat{P})$:
  - If $\hat{B}$ is protected by the TPM, $\mathcal{S}_7$ simulates this session activation with the help of $\mathsf{ephem}_B()$.
  - If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_7$ simulates this session activation as follows with the help of a Table T':
    1) Randomly choose $e, s \in Z_q$.
    2) Set $Y = g^s/B^e$ and $e = avf(Y)$.
    3) Add an entry $(e, s, Y, -)$ to T'.
    4) Output Y.
- Respond$(sc, \hat{B}, \hat{P}, X)$:
  1) If $\hat{B}$ is protected by the TPM, $\mathcal{S}_7$ simulates this session activation with the help of $\mathsf{ephem}_B()$ and $\mathcal{O}_{\hat{B}}^{\mathsf{MQV}}$.
  2) If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_7$ simulates this session activation as follows:
     a) Randomly choose $e, s \in Z_q$.
     b) Set $Y = g^s/B^e$ and $e = avf(Y)$.
     c) Compute $Z_1 = (XP^d)^{hs}$ where $d = avf(X)$.
     d) Output Y, and compute the session key $k = H_2(Z_1, \hat{B}, \hat{P})$.
- Complete$(sc, \hat{B}, \hat{P}, Y, X)$:
  1) If $\hat{B}$ is protected by the TPM, $\mathcal{S}_7$ simulates this session activation with the help of $\mathsf{ephem}_B()$ and $\mathcal{O}_{\hat{B}}^{\mathsf{MQV}}$.
  2) If $\hat{B}$ is not protected by the TPM, $\mathcal{S}_7$ proceeds as follows:
     a) Check whether (1) $sc =$ TPM_ALG_ECMQV, (2) P and X are on the curve associated with B, and (3) the last element of the record in T' indexed by Y is '$-$'. If the above checks succeed, continue, else return an error.
     b) Suppose the entry in T' indexed by Y is $(e, s, Y, -)$, set $Z_1 = (XP^d)^{hs}$ where $d = avf(X)$, set the last element of the entry to be $\times$, and compute the session key $k = H_2(Z_1, \hat{B}, \hat{P})$.
- SessionStateReveal(s): $\mathcal{S}_7$ returns $\mathcal{M}$ the session key of s. Besides, $\mathcal{S}_7$ also returns the 3-tuple $\sigma = (Z-1, \hat{P}_1, \hat{P}_2)$. The reason that $\mathcal{S}_7$ can return the 3-tuple σ of s to $\mathcal{M}$ is that in the above simulation of session activations at $\hat{B}$, we compute the 3-tuple σ of each session, and for other parties, $\mathcal{S}_7$ is also able to compute the 3-tuple σ of each session.

Proof: As $\mathcal{S}_7$ provides $\mathcal{M}$ with the 3-tuple σ of all exposed sessions, $\mathcal{M}$ can query s' to obtain the 3-tuple σ' of s' which equals σ of the test session s. This means that $\mathcal{M}$ successfully gets the 3-tuple of the test session s without exposing s or its matching session, namely, $\mathcal{M}$ can launch the forging attack. We have proved that if $\mathcal{M}$ can launch forging attacks, we can build a GDH solver $\mathcal{S}_6$. So $\mathcal{S}_7$ can leverage $\mathcal{S}_6$ to solve the CDH problem: $CDH(X_0, B) = \mathcal{S}_6(X_0, B)$. ■

3) Analysis of Cases 3: The analysis of Case 3 is similar to Case 2, and it can be easily completed following the analysis of Case 2.

XIII. Further Security Properties of tpm.KE.rev

In this section, we analyze the KCI-resistance and weak PFS (wPFS) security properties of tpm.KE.rev.

A. Resistance to KCI Attacks

The KCI-resistance property means that even if the attacker $\mathcal{M}$ has obtained the long-term key of party $\hat{A}$, he cannot impersonate other parties to $\hat{A}$. We show that the Full UM protocol cannot achieve this security property by the following attack: $\mathcal{M}$ generates an ephemeral key pair $(y, Y = g^y)$ and initiates a session with $\hat{A}$ as the identity of $\hat{B}$; after receiving Y, $\hat{A}$ generates its ephemeral key pair $(x, X = g^x)$ and computes its session key $k = H_1(B^a, Y^x, \hat{A}, \hat{B}, X, Y)$; $\mathcal{M}$ can also compute the session key of $\hat{A}$: $k = H_1(B^a, X^y, \hat{A}, \hat{B}, X, Y)$. We now prove that the MQV and SM2 key exchange protocols of tpm.KE.rev can achieve the KCI-resistance property.

Lemma 9: Under the GDH assumption, the MQV and SM2 key exchange protocols of tpm.KE.rev resist KCI attacks.

Proof: Lemma 9 can be easily proved by slightly modifying the proof of MQV and SM2 key exchange protocols in Section XII. The only change to the proof in Section XII is that all GDH solvers used to prove the security of MQV and SM2 key exchange protocols do not abort when $\mathcal{M}$ corrupts $\hat{A}$. The proof remains valid because the above abort operations are not used in the proof (we add the abort of GDH solvers when $\hat{A}$ is corrupted for compliance with the notion of the session exposure in the security model). ■

B. Weak Perfect Forward Secrecy

Hrawczyk has proved that the implicitly AKE protocols cannot achieve the full PFS property and can only achieve weak PFS [21]: only the session keys that are established without the active involvement of the attacker enjoy PFS property.

Lemma 10: Under the CDH assumption, tpm.KE.rev provides weak PFS.

Proof: Here we only outline the idea of the proof of Lemma 10, and the full proof can be completed following the proof in Section XII. We construct a CDH solver $\mathcal{S}_8$ if $\mathcal{M}$ successfully breaks the weak PFS security property. Let X and Y be the inputs to $\mathcal{S}_8$, and the goal of $\mathcal{S}_8$ is to compute $CDH(X, Y)$. $\mathcal{S}_8$ simulates the environment for $\mathcal{M}$ as follows: it sets all parties' long-term keys and chooses a random guess for the test session of the distinguishing game. We call the guessed session the g-session, denote the owner and the peer of the test session by $\hat{A}$ and $\hat{B}$ respectively, and denote their long-term private keys by a and b, respectively. $\mathcal{S}_8$ sets the incoming and outgoing messages in the g-session to be X and Y. If $\mathcal{M}$ does choose the g-session as the test session and wins the distinguish game, he must compute the tuple σ of the test session in his run. We proceed to show that no matter what the type of A is, $\mathcal{S}_8$ can compute $CDH(X, Y)$, i.e., g^{xy}.

1) Case TPM_ALG_ECDH: σ equals $(Z_1, Z_2, \hat{A}, \hat{B}, X, Y)$ where $Z_2 = g^{xy}$.
2) Case TPM_ALG_ECMQV: σ equals $(Z, \hat{A}, \hat{B})$ where $Z = g^{h(x+da)(y+eb)}$; since $\mathcal{S}_8$ knows a and b, he can compute g^{xy}.
3) Case TPM_ALG_SM2: σ equals $(Z, \hat{A}, \hat{B})$ where $Z = g^{h(a+dx)(b+ey)}$; since $\mathcal{S}_8$ knows a and b, he can compute g^{xy}.

■

XIV. CONCLUSION

In this paper, we present a formal analysis of the secure communication interfaces of TPM 2.0 in a unified way. We construct a security model which takes account of the protections of the TPM on keys and protocols' computation environments and eliminates group representation attacks existing in theory by measuring the entropy of the output of the $avf()$ and $avf'()$ functions. The analysis results show that the current version of the key exchange primitive in TPM 2.0 can achieve the basic security property of modern security models, but its security is achieved under some impractical conditions: all devices in the network should be protected by the TPM, and security measures should be deployed to prevent sophisticated physical attacks, which can be used by attackers to obtain keys inside the TPM. Besides, the analysis results also show that the current version of the key exchange primitive in TPM 2.0 cannot achieve other security properties, such as KCI-resistance and weak PFS properties. We also give suggestions to engineers on how to use the key exchange primitive of TPM to implement key exchange protocols securely.

To eliminate the impractical conditions required by the TPM 2.0, we revise the key exchange primitive of TPM 2.0 and give a rigorous analysis of the revision. The analysis results show that our revision helps the key exchange primitive of TPM 2.0 not only enjoy the essential security property defined by modern AKE models in real-world networks but also achieve additional security properties: the MQV and SM2 key exchange protocols of TPM 2.0 enjoy the KCI-resistance property, and all the three protocols of TPM 2.0 enjoy the weak PFS property. Our revision only needs to modify one TPM command, and we give concrete suggestions on how to revise the TPM 2.0 specifications.

APPENDIX 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 to 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) $\mathcal{M}$ 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.

APPENDIX 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, $\mathcal{M}$ needs 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.

A. Attack I

$\mathcal{M}$ selects $u \in_R Z_q$, and registers a carefully 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 $(\hat{B}, \mathcal{M}, Y, X)$, and then $\mathcal{M}$ can compute Z_A of session $(\hat{A}, \hat{B}, X, Y)$: $Z_A = Z_B/(BY^e)^{hu}$, and $\mathcal{M}$ further derives the session key of $(\hat{A}, \hat{B}, X, Y)$.

Note that the above attack shows that the 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. 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 $(\hat{B}, \mathcal{M}, Y, X')$, 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 $(\hat{A}, \hat{B}, X, Y)$.

APPENDIX 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 representations of its elements satisfy that the $\lceil q/2 \rceil$ LSBs of the representation of points' x-coordinate are fixed. We use c to denote the 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 $K = H_2((YB^c)^{hx^*}, \hat{A}, \hat{B})$.

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

REFERENCES

[1] GM/T 0003: Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves, Part 3: Key Exchange Protocol.
[2] ISO/IEC 15946-5: Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 5: Elliptic curve generation.
[3] William Aiello, Steven M Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Angelos D Keromytis, and Omer Reingold. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):242–273, 2004.
[4] American National Standard (ANSI) X9.42. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography, 2003.
[5] American National Standard (ANSI) X9.63. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography, 2001.
[6] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.
[7] ARM. Security Technology - Building a Secure System using Trustzone Technology. *ARM Technical White Paper*, 2009.
[8] Elaine B Barker, Don Johnson, and Miles E Smid. NIST SP 800-56A. Recommendation for Pair-wise Key Establishment Schemes using Discrete Logarithm Cryptography (Revised). 2007.
[9] Simon Blake-Wilson and Alfred Menezes. Unknown key-share attacks on the station-to-station (sts) protocol. In *Public Key Cryptography*, pages 154–170. Springer, 1999.
[10] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145. ACM, 2004.
[11] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – EUROCRYPT 2001*, pages 453–474. Springer, 2001.

[12] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *Advances in Cryptology – CRYPTO 2002*, pages 143–161. Springer, 2002.

[13] Lily Chen. Recommendation for key derivation using pseudorandom functions. *NIST Special Publication*, 800:108, 2008.

[14] Liqun Chen and Bogdan Warinschi. Security of the TCG Privacy-CA solution. In *In 2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 609–616. IEEE, 2010.

[15] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[16] FIPS, PUB. 186-2. Digital Signature Standard (DSS). *National Institute of Standards and Technology (NIST)*, 2000.

[17] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin. Okamoto-Tanaka revisited: Fully authenticated Diffie-Hellman with minimal overhead. In *Applied Cryptography and Network Security*, pages 309–328. Springer, 2010.

[18] ISO/IEC IS 15946-3. Information Technology - Security Techniques - Cryptographic Techniques Based on Elliptic Curves - Part 3: Key Establishment, 2002.

[19] Ik Rae Jeong, Jonathan Katz, and Dong Hoon Lee. One-round protocols for two-party authenticated key exchange. In *Applied Cryptography and Network Security*, pages 220–232. Springer, 2004.

[20] Burton 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.

[21] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology – CRYPTO 2005*, pages 546–566. Springer, 2005.

[22] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16. Springer, 2007.

[23] Kristin Lauter and Anton Mityagin. Security analysis of KEA authenticated key exchange protocol. In *Public Key Cryptography – PKC 2006*, pages 378–394. Springer, 2006.

[24] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[25] Tsutomu Matsumoto and Youichi Takashima. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.

[26] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. *HASP@ ISCA*, 10, 2013.

[27] Alfred Menezes. Some New Key Agreement Protocols Providing Implicit Authentication. In *Second Workshop on Selected Areas in Cryptography, 1997*. CRC Press, 1997.

[28] ORACLE. Java Card Application Programming Interface (API) Specification Version 3.1. https://docs.oracle.com/en/java/javacard/3.1/specnotes/index.html, 2019.

[29] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology – EUROCRYPT'96*, pages 387–398. Springer, 1996.

[30] Jordan Robertson. Security chip that does encryption in PCs hacked. *USA Today, Feb*, 8, 2010.

[31] SECG. Recommended Elliptic Curve Domain Parameters. *http://www.secg.org*, 2000.

[32] Skipjack and NIST. KEA algorithm specifications, 1998.

[33] Standardization Administration of the People's Republic of China. GB/T 32918.3-2016. Information security technology - Public key cryptographic algorithm SM2 based on elliptic curves - Part 3: Key exchange protocol (in Chinese), 2016.

[34] TCG. TPM Demonstrated as Key Element of Industries 4.0 at German IT Summit. https://trustedcomputinggroup.org/tpm-demonstrated-key-element-industries-4-0-german-summit, 2015.

[35] TCG. Trusted Platform Module Library Part 1: Architecture, Family 2.0, Level 00 Revision 01.38, 2016.

[36] TCG. Trusted Platform Module Library Part 3: Commands Family 2.0, Level 00 Revision 01.38, 2016.

[37] TCG. Protection Profile Automotive - Thin Specific TPM for TCG TPM 2.0 Automotive Thin Profile Family 2.0 Level 0, 2018.

[38] TCG. TCG Algorithm Registry Family 2.0, Level 00 Revision 01.27, 2018.

[39] TCG. TCG TPM 2.0 Automotive Thin Profile - For TPM Family 2.0, Level 0 Revision 15, 2018.

[40] Berkant Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Designs, Codes and Cryptography*, 46(3):329–342, 2008.

[41] Frans MJ Willems, Yuri M Shtarkov, and Tjalling J Tjalkens. The context-tree weighting method: Basic properties. *Information Theory, IEEE Transactions on*, 41(3):653–664, 1995.

[42] Jing Xu and Dengguo Feng. Comments on the SM2 key exchange protocol. In *Cryptology and Network Security*, pages 160–171. Springer, 2011.

[43] Andrew C Yao and Yunlei Zhao. A New Family of Implicitly Authenticated Diffie-Hellman Protocols. Technical report, 2011.

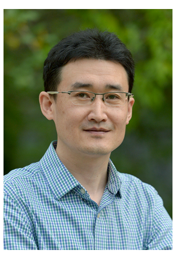
[44] Andrew Chi-Chih Yao and Yunlei Zhao. OAKE: A New Family of Implicitly Authenticated Diffie-Hellman Protocols. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1113–1128. ACM, 2013.

[45] Shijun Zhao, Li Xi, Qianying Zhang, Yu Qin, and Dengguo Feng. Security analysis of SM2 key exchange protocol in TPM2. 0. *Security and Communication Networks*, 8(3):383–395, 2015.

[46] Shijun Zhao and Qianying Zhang. A Unified Security Analysis of Two-phase Key Exchange Protocols in TPM 2.0. In *International Conference on Trust and Trustworthy Computing*, pages 40–57. Springer, 2015.

Qianying Zhang received her Ph.D. degree from Institute of Software, Chinese Academy of Sciences in 2015. She is currently a lecturer at Capital Normal University. Her research interests include cryptography, information security, operating system security, and formal verification.

Shijun Zhao received his Ph.D. degree from Institute of Software, Chinese Academy of Sciences in 2015. He is currently an associate researcher at Institute of Software, Chinese Academy of Sciences. His research interests include cryptography, information security, operating system security, and mobile device security.



Zhiping Shi received his Ph.D. degree in computer software and theory at Institute of Computing Technology, Chinese Academy of Sciences in 2005. He is currently a Professor at Capital Normal University, Beijing, China. His research interests include formal methods and theorem proving.

Yong Guan received his Ph.D. degree from the College of Mechanical Electronic and Information Engineering, China University of Mining and Technology, Xuzhou, China, in 2004. He is currently a professor at Capital Normal University, Beijing, China. His research interests include formal verification of embedded system design. Dr. Guan is a member of the Chinese Institute of Electronics Embedded Expert Committee.

Guohui Wang received his BE degree in electronic information engineering from Chongqing University of Technology, China, in 2007 and the ME degree in communications and information technology from Capital Normal University, Beijing, China, in 2011. He is currently working toward the Ph.D. degree in the School of Mathematical Sciences, Capital Normal University, Beijing, China. His main research interests include embedded system design and formal verification.