A TPM Diffie-Hellman Oracle

Tolga Acar Lan Nguyen Greg Zaverucha
Microsoft Research, Redmond, WA

October 18, 2013*

Abstract

This note describes a Diffie-Hellman oracle, constructed using standard Trusted Platform Module (TPM) signature APIs. The oracle allows one to compute the exponentiation of an arbitrary group element to a specified TPM-protected private key. By employing the oracle, the security provided by a group of order p is reduced by $\log k$ bits, provided k oracle queries are made and $p \pm 1$ is divisible by k. The security reduction follows from a straightforward application of results from Brown and Gallant (IACR ePrint 2004/306) and Cheon (Eurocrypt 2006) on the strong Diffie-Hellman problem. On a more positive note, the oracle may allow a wider range of cryptographic protocols to make use of the TPM.

1 Introduction

Trusted Computing Group (TCG) recently published an updated draft for Trusted Platform Module (TPM) Family 2.0 [1]. For convenience, we loosely call this TPMv2. TPMv2 allows, amongst other things, the generation of DSA and ECDSA key pairs with unextractable private keys. Such private keys never leave the TPM security boundary unencrypted. TPMv2 also added ECDAA (ECC-based Direct Anonymous Attestation) support along with a version of elliptic curve Schnorr signatures to handle protocols like U-Prove [3, 8]. The Schnorr signature generation is possible by using two TPM commands: TPM_Commit and TPM_Sign [2]. Depending on inputs to the TPM_Commit command, the resulting signature is either exactly Schnorr's scheme, or a variant of the scheme where a different group element is used in signature generation.

We show that this capability provides a *TPM Diffie-Hellman Oracle*: Given any arbitrary group element input, TPMv2 allows exponentiation of the group element to a specified TPM private key. This is possible because the TPMv2 commands can actually do more than just generate Schnorr signatures. More specifically, the TPM_Commit command allows committing to a random value r as g^r using any group element g, whereas the

*Revised October 23, 2013.

Schnorr signature scheme specifies that g must be the base element used to generate the signer's public key (usually the generator of the group. We exploit this to help construct the oracle.

The DH oracle has both constructive and destructive aspects. On one hand, it potentially reduces the complexity of recovering TPM-protected DSA and ECDSA private keys. Previously, it was expected that recovering these keys would be as difficult as solving an instance of the discrete logarithm problem (DLP) (i.e., parameters are chosen to ensure the difficulty of the DLP). We note that the reduced complexity is still asymptotically exponential, which means that these TPM keys could still be secure with larger security parameters. On the other hand, as we can now exponentiate any base to a TPMv2 private key, more crypto protocols can be potentially integrated with TPMv2. This increases the possibility of adding hardware-supported security to more crypto protocols (in addition to those explicitly supported by TPM APIs), as TPMs are becoming available in more devices.

We have reported this finding to TCG. TCG has updated the TPMv2 standard to limit the use of the TPM_Commit and TPM_Sign functions; they can now only be used with keys explicitly flagged for use with certain protocols (anonynimity protocols such as ECDAA).

2 TPM APIs for Schnorr Signatures

We consider the following TPMv2 API commands with Schnorr signatures: TPM_Create, TPM_CreatePrimary, TPM_Commit, and TPM_Sign. A TPM ECC signing key can be used with a number of schemes, including but not limited to ECDSA, ECDAA, ECSCHNORR, and SM2. Thus, the oracle is applicable to TPMv2 ECC signing keys and those schemes. (Note that, SM2 algorithm does not need to use the TPM_Commit command.) TPMv2 offers the two-command signature with TPM_Commit and TPM_Sign (as opposed to a single commnad) to enable anonymity protocols such as ECDAA. For TPMv2 keys used with more common signature algorithms, the TPM_Sign command generates a signature without the TPM_Commit command [2]. In the following descriptions, TPMv2 commands accept a handle of an appropriate key.

Parameters. The scheme is defined with respect to a cyclic group $\mathbb{G}$ of prime order p, and a hash function H.

Key Generation. The TPM generates a TPM private key $x \in_R \mathbb{Z}_p$, and computes the public key $y := g^x$, using the TPM_Create or TPM_CreatePrimary commands. The public key y is output, and the private key x never leaves the TPM unencrypted.

Signature Generation. The signature generation uses the ECDAA scheme on TPMv2. The TPM_Commit command computes $W := g^w$ for a random w, generated internally, and kept private on the TPM. The TPM_Sign command accepts an arbitrary

challenge c to compute $r = cx + w \pmod{p}$. The signature output is (c, r). TPMv2 maintains w internally across the TPM_Commit and TPM_Sign calls. (This is done by using a counter, TPM-internal secret, and a KDF. The details are not important for this analysis; only the effect that the same random number is used in TPM_Commit and TPM_Sign commands). In order to create a valid Schnorr signature, c must be $H(W||m)$ for an arbitrary message m.

Signature Verification. Verification is done externally, without any TPM commands. To verify that (c, r) is a valid signature on m, check that $c = H(g^r/y^c||m)$.

3 Construction of a Diffie-Hellman Oracle

This section describes the new TPM oracle, which is a DH oracle. The term *static DH oracle* was first used by Brown and Gallant [4] to describe a function, usually in the context of a cryptographic protocol, which leaks a DH value h^x, where $h \in \mathbb{G}$ is an arbitrary input value and x is a secret. Our goal is to use the TPMv2 APIs to define a function $O_x : \mathbb{G} \to \mathbb{G}$ such that $O_x(h) = h^x$, where $(x, y := g^x)$ is a TPM protected key pair. The following steps define the TPM oracle O_x based on the commands described in Section 2.

DH Oracle O_x

1. Call TPM_Commit with input h. The output is $W := h^w$ for a random $w \in \mathbb{Z}_p$.
2. Call TPM_Sign with an arbitrary input c. The output is $r := cx + w$.
3. Compute and output $(h^r/W)^{1/c} = h^x$.

4 Implications

We first recognize an observation on page 8 of [4] in the context of smart cards:

> *Some systems using hardware modules for protecting private keys might provide an SDHP oracle to an adversary. For example, a smart card is a highly constrained environment, and sometimes all hashing and key derivation is done outside the security boundary of the smart card, usually on the smart card reader. A smart card (holding a user's private key q) used in an encryption scheme may be presented with a recipient public key R and the card may simply return the element qR to the reader, where the subsequent encryption processing is performed. In this case, a malicious smart card reader could use the smart card as an SDHP oracle.*

Our observation confirms the existence of an SDHP oracle in TPMv2, though not as direct as the smart card case (which 'simply returns the element qR'). In our context, the TPM is the relatively constrained environment providing the oracle.

4.1 Security Analysis

We use the complexity analysis of the Strong Diffie-Hellman (SDH) problem of Brown and Gallant [4], and Cheon [5]. The k-SDH problem is to compute x, given $g, g^x, g^{x^2}, \ldots, g^{x^k}$. The TPMv2 DH oracle described here can be used to compute this sequence, an instance of the SDH problem. Then we can use the SDH algorithms of [4, 5] to recover x, faster than had we been given only g^x, as is normally the case with Schnorr or ECDSA key pairs. The complexity of the Cheon SDH algorithm is $O(2^{(\ell_p - \ell_k)/2})$ where ℓ_p is the bit length of p and ℓ_k is the bitlength of k. By contrast, the best known algorithm for recovering x given g^x in an elliptic curve group requires $O(2^{\ell_p/2})$ work.

For example, when G is an elliptic curve group with 256-bit prime order and if $2^{20} \approx$ 1M TPMv2 oracle queries are allowed, then the expected security of the TPM protected secret would decrease from 2^{128} to 2^{118}.

A few remarks are in order.

- TPMs can export secret key material encrypted for import to another TPM. In this case, multiple TPMs may be queried to construct the SDH instance.
- For the Cheon and Brown-Gallant algorithms to work, there is a technical condition on the group order p: k must divide $p \pm 1$. In practice, since groups are typically not chosen to constrain $p \pm 1$, there are often many divisors available. In [4], a table of factorizations of $p - 1$ are given for the NIST curves. All of them have sufficiently many small divisors to enable the attack to some degree.
- The TPM queries must be made sequentially. As far as we know, g^{x^i} is required in order to compute $g^{x^{i+1}}$ with the DH oracle described above.

4.2 Protocol Integration with the TPM

There have been a number of works on enhancing security of existing crypto protocols using hardware [6, 7]. Given that TPM can now be used to perform exponentiation of any base to the private key, and exponentiation is a common way to hide secrets, this DH oracle could allow many cryptographic protocols to take advantage of TPMv2 capabilities. Integration with U-Prove is one example [8].

5 Acknowledgments

We would like to thank David Wooten (Microsoft), Christian Paquin (Microsoft), Liqun Chen (HP) and Jiangtao Li (Intel) for TPMv2 intricacies and comments.

References

[1] TCG Public Review. Trusted Platform Module Library. Part 1: Architecture. Family 2.0. August 22, 2013, Committee Draft, Level 00 Revision 00.99.

[2] TCG Public Review. Trusted Platform Module Library. Part 3: Commands Family 2.0. August 22, 2013, Committee Draft, Level 00 Revision 00.99.

[3] C.P. Schnorr. Efficient Identification and Signatures for Smart Cards. Proceedings of CRYPTO'89, pp.239-252, LNCS.435, 1989.

[4] D. Brown and R. Gallant. The Static Diffie-Hellman Problem. IACR ePrint Report 2004/306, November 2004.

[5] J. Cheon. Security Analysis of the Strong Diffie-Hellman Problem. In *Proceedings of EUROCRYPT 2006*.

[6] M. Fischlin, B. Pinkas, A. Sadeghi, T. Schneider and I. Visconti. Secure Set Intersection with Untrusted Hardware Tokens. In *Proceedings of CT-RSA 2011*.

[7] C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *Proceedings of ACM CCS 2008*.

[8] C. Paquin, G. Zaverucha. U-Prove Cryptographic Specification V1.1 (Revision 2). Microsoft Research Technical Report, April 2013.