RSA Signatures Under Hardware Restrictions

Marc Joye
NXP Semiconductors
San Jose, CA, USA
marc.joye@nxp.com

Yan Michalevsky
Anjuna Security
Palo Alto, CA, USA
yanm2@cs.stanford.edu

ABSTRACT

We would like to compute RSA signatures with the help of a Hardware Security Module (HSM). But what can we do when we want to use a certain public exponent that the HSM does not allow or support? Surprisingly, this scenario comes up in real-world settings such as code-signing of Intel SGX enclaves. Intel SGX enclaves have to be signed in order to execute in release mode, using 3072-bit RSA signature scheme with a particular public exponent. However, we encountered commercial hardware security modules that do not support storing RSA keys corresponding to this exponent.

We ask whether it is possible to overcome such a limitation of an HSM and answer it in the affirmative (under stated assumptions). We show how to convert RSA signatures corresponding to one public exponent, to valid RSA signatures corresponding to another exponent. We define security and show that it is not compromised by the additional public knowledge available to an adversary in this setting.

CCS CONCEPTS

• **Security and privacy → Digital signatures**; **Tamper-proof and tamper-resistant designs**; • **Hardware → Communication hardware, interfaces and storage**;

KEYWORDS

Digital Signatures; RSA; Hardware Security; Hardware Security Modules; Secure Enclaves.

ACM Reference Format:
Marc Joye and Yan Michalevsky. 2018. RSA Signatures Under Hardware Restrictions. In *The Second Workshop on Attacks and Solutions in Hardware Security (ASHES'18), October 19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3266444.3266451

1 INTRODUCTION

We examine a peculiar but realistic problem. We would like to compute RSA signatures with the help of a Hardware Security Module (HSM).[1] But what can we do when we want to use a certain public exponent e that the HSM does not allow or support? Surprisingly, this scenario comes up in real-world settings.

One setting in which we encounter this problem is related to Intel's implementation of secure enclaves. Secure enclaves are execution units that are isolated from any other code running at similar or higher privilege levels. Such isolation, along with attestation of the enclave's state, effectively provides a trusted execution environment (TEE) inside the CPU. Intel realized this concept as part of Software Guard Extensions (SGX) that were introduced with the Skylake architecture [13].

Intel SGX enclaves have to be signed in order to run in production mode. The enclave signature is verified by an architectural enclave called the Launching Enclave (LE). The LE checks that the public key is approved and whitelisted by Intel, and uses it to verify the enclave signature. If verification passes it would execute the enclave, or reject it otherwise. It is currently mandatory that enclaves are signed using 3072-bit RSA with public exponent $e = 3$.

An organization that would like to secure the private key used to sign its enclaves would often turn to Hardware Security Modules (HSM) as a solution for securely storing and using cryptographic keys. Moreover, such precaution is in fact recommended by Intel in its signing and whitelisting instructions for Intel SGX enclaves [14]. However, not all HSMs enable arbitrary parameters.

1.1 Background

The choice of a small public exponent enables faster and less computationally intensive verification of RSA signatures. This could be especially important for small low-power devices with modest processing budget.

While there are practical attacks on encryption and signing using low *private* exponent [18], there are no practical attacks that break a correct implementation of RSA encryption or signing using a small *public* exponent. Boneh [6] provides a useful survey of the main attacks that emerged after years of research.

A well-known theoretic attack on RSA encryption and signing using a low public-exponent is due to Coppersmith's theorem [6, 9]. Coppersmith's LLL-based technique provides an efficient algorithm for finding small roots of polynomial equations modulo N. Shimizu [17] applied this algorithm to an old attack due to Håstad [12] for recovering a message encrypted to multiple recipients, each having their own RSA public key, when padding is either not used or done improperly. Franklin et al. [10] attack the encryption of related messages under the same modulus. Coppersmith proposes an improved attack on encryption with short padding [9]. However, those attacks are mitigated by proper padding of the encrypted message, and use of standard and widely-used cryptographic libraries easily prevents them.

Boneh et al. [7] showed that if an attacker is able to somehow recover some of the bits of the private key, she would be able to recover the entire private key if the public exponent is low ($e < \sqrt{N}$). It is therefore important to safeguard the entire RSA private key.

[1] Or some other cryptographic key-management solution.

ASHES'18, October 19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5996-2/18/10…$15.00
https://doi.org/10.1145/3266444.3266451

In practice, attackers are likely to either fail to obtain access to the private key whatsoever, or manage to get all its bits. Some cases where the concern about partial bits exposure makes sense is side-channel attacks such as power-analysis or cache-timing attacks where the attacker might be able to read some bits with reasonable probability, but encounter noise that makes it somewhat harder (or more time consuming) to extract all bits of the key. In this case, using a larger exponent can make it harder for the attacker to succeed.

Therefore, despite the arguments in favor of using RSA with small public exponents (given sound message padding such as in RSAES-OAEP or RSASSA-PSS, or Schemes 2 or 3 from ISO/IEC 9796-2), vendors may choose to restrict support to larger exponents. For example, Yubico's YubiHSMv2 does not allow storage of RSA signing keys with public exponent $e = 3$.[2] Yubico replied to our inquiry about the lack of support for this exponent and stated that:

> *"While there are no conclusive proofs of its unsuitability, the use of public exponent $e = 3$ is discouraged, especially when used in conjunction with padding schemes like PKCS#1 v1.5. We chose not to add support for other exponents than $F_4 = 2^{16} + 1$ which is the de-facto industry standard."*

1.2 Our Contribution

Our goal is to allow using such restricted equipment while enabling to use RSA exponents unsupported by the hardware. In our setting we have short term access to the private key, that enables us to compute a certain transformation. Once we compute it we upload the new private key obtained via the transformation to a secure storage such as a HSM, and "forget" it. From that point on, we no longer have access to the private key.

1.3 Related Work

Our setting is similar to proxy re-signatures, first introduced by Blaze et al. [5], and later revisited by Ateniese and Hohenberger [1]. In this setting, a semi-trusted proxy is used to convert a signature from Alice to a signature from Bob on the same message, without the need for the proxy to possess Alice's or Bob's private keys. However, these works assume that both public keys are known, and therefore realization with RSA using a common modulus N, and public e and e', are not secure. We work around the issue of having a proxy re-signature scheme that translates between two publicly verifiable RSA signatures sharing the same modulus, by *not requiring* one of the signatures to be publicly verifiable.

2 KEY TRANSFORM

We start by recalling the general construction of the RSA-based signature schemes. We use a general padding function

$$\mu\colon \mathcal{M} \to \mathbb{Z}_N, m \mapsto \mu(m)$$

where $\mathcal{M}$ denotes the message space. Commonly used padding schemes include Full-Domain Hash (FDH) [2] and Probabilistic Signature Scheme (PSS) [4].

[2]Our proposal does not apply in the particular case of YubiHSMv2 since they are limited to a single public exponent ($e = 2^{16} + 1$), but it can be helpful when certain public exponents, or a certain range, are prohibited.

Definition 2.1 (RSA-based signature scheme). The signature scheme is given by three algorithms:

KeyGen(k) On input of a security parameter k, the algorithm outputs a public key $pk = \{N, e\}$ where N is a composite integer of unknown factorization, and a private key $sk = \{d\}$, such that[3] $e \cdot d \equiv 1 \pmod{\varphi(N)}$.

Sign(N, d, m) Output $\sigma = \mu(m)^d \bmod N$.

Verify(N, e, m) If $\mu(m) \equiv \sigma^e \pmod N$, output *true*; otherwise output *false*.

2.1 Construction

Let e be the actual exponent we want to use. We generate a public modulus N as a multiple of primes (as prescribed for RSA), pick a *random* exponent $t \xleftarrow{R} \mathcal{T}$, where

$$\mathcal{T} = \left\{ t \in \mathbb{Z} \mid 0 < t < N \text{ and } \gcd(t, \varphi(N)) = 1 \right\} .$$

We compute $e' = e \cdot t \bmod \varphi(N)$, and check that e' is supported by the HSM. Finally, we compute

$$d' = (e')^{-1} \bmod \varphi(N)$$

and store d' in the HSM. As will become apparent in Sect. 3.1, **it is mandatory that e' is not made public, and that it has sufficient entropy**.

On input a message m, the HSM issues an RSA signature $\sigma' = \mu(m)^{d'} \bmod N$.

Remark 1. It is worth noting that (e', d') is a valid RSA key pair. By definition, the value of t satisfies $\gcd(t, \varphi(N)) = 1$. Hence, since $e \in \mathbb{Z}^*_{\varphi(N)}$, it follows that $e' = e \cdot t \bmod \varphi(N) \in \mathbb{Z}^*_{\varphi(N)}$ and, consequently, $d' = (e')^{-1} \bmod \varphi(N) \in \mathbb{Z}^*_{\varphi(N)}$.

Integer t is called the *key-transform exponent* and is public. An HSM-generated signature σ' can be publicly converted into an original signature σ as

$$\sigma = (\sigma')^t \bmod N \ .$$

It is easily verified that $(\sigma')^t \equiv \left[\mu(m)^{d'}\right]^t \equiv \left[\mu(m)^{\frac{1}{e'}}\right]^t \equiv \mu(m)^{\frac{1}{e}} \equiv \mu(m)^d \equiv \sigma \pmod N$.

We prove in the next section that the additional knowledge of t does not compromise the security of the scheme. It inherits the security guarantees as the original scheme.

3 SECURITY ANALYSIS

3.1 A Key Recovery Attack

We show that it is important to keep e' secret. Indeed, suppose that someone were able to retrieve the value of e'. Then, assuming that $\gcd(e, e') = 1$, the extended Euclidean algorithm would yield integers a and b such

$$a \cdot e + b \cdot e' = 1 \ .$$

In turn, since $e' = e \cdot t \bmod \varphi(N)$, this would imply $1 \equiv a \cdot e + b \cdot e \cdot t \equiv e\,(a + b \cdot t) \pmod{\varphi(N)}$ and thus

$$D := a + b \cdot t \equiv e^{-1} \equiv d \pmod{\varphi(N)} \ .$$

[3]RSA exponents are defined modulo Euler's totient function, $\varphi(N)$. They can similarly be defined modulo Carmichael's function, $\lambda(N)$.

The adversary can now produce a valid signature on a message m by computing

$$\mu(m)^D \equiv \mu(m)^{a+bt} \equiv \mu(m)^d \pmod{N} \ .$$

This is essentially equivalent to the statement in [1] that a proxy re-signature scheme cannot translate between two publicly-verifiable RSA signatures sharing the same modulus.

3.2 Key Transform Security

We prove by simulation that the construction given in Sect. 2 is existentially unforgeable against chosen-message attacks (EUF-CMA), reducing to the RSA assumption in the random-oracle model. For concreteness, we instantiate the signature scheme with RSA-FDH. In this case, the padding μ is defined by a cryptographic hash function $\mathcal{H}\colon \mathcal{M} \to \mathbb{Z}_N$. Extensions to other padding schemes are discussed in Sect. 4.

Assumption 1 (RSA). *Given a modulus N of unknown factorization, an integer $e \in \mathbb{Z}^*_{\varphi(N)}$ and a randomly sampled integer $y \xleftarrow{R} \mathbb{Z}_N$, a polynomial-time adversary has negligible probability of finding x such that $x^e \equiv y \pmod{N}$.*

We assume that there exists an EUF-CMA adversary $\mathcal{A}$ against the proposed signature scheme, that returns a signature forgery with probability $\epsilon_{\mathcal{A}}$ and within polynomial time $\tau_{\mathcal{A}}$. We will use this adversary to solve a given instance of the RSA problem with success probability ϵ and within polynomial time τ.

RSA Challenge. On input an RSA public key $\{N, e\}$ and an integer $y \xleftarrow{R} \mathbb{Z}_N$, the goal is to find $x \in \mathbb{Z}_N$ such that $y \equiv x^e \pmod{N}$.

Simulation. We describe below the simulation of the key generation algorithm, the hash oracle (random oracle) and the signing oracle. A history list Hist[$\mathcal{H}$] is maintained. It will consist of tuples of the form $(m_i, r_i, h_i, b_i) \in \mathcal{M} \times \mathbb{Z}_N \times \mathbb{Z}_N \times \{0, 1\}$; Hist[$\mathcal{H}$] is initially set to $\emptyset$. At the beginning of the simulation, the reduction algorithm selects a subset $\mathcal{L}$ of ℓ (different) integers, $\mathcal{L} = \{j_1, \ldots, j_\ell\}$, at random in the set $\{1, \ldots, q_H + q_S\}$. It also sets a counter i to 0.

q_H denotes the number of hash queries that are not later followed by a signature query on the same message; q_S denotes the number of signature queries.

Key generation Public RSA exponent and modulus are set by the values (e, N) given in the RSA challenge. An odd integer t is chosen at random in $[1, N)$. The corresponding public key $pk = \{e, N\}$ and transform-exponent t are given to $\mathcal{A}$.

Random oracle Let m denote an input message to hash function $\mathcal{H}$, modeled as a random oracle.

- If $m \in$ Hist[$\mathcal{H}$] then h_i is returned as the hash value of m, $\mathcal{H}(m) := h_i$, where (m, r_i, h_i, b_i) is the entry in Hist[$\mathcal{H}$] corresponding to m;
- If $m \notin$ Hist[$\mathcal{H}$] then counter i is incremented, $i \leftarrow i + 1$, and
  - If $i \notin \mathcal{L}$ then h_i is defined as $h_i = {r_i}^{et} \bmod N$ for a random integer $r_i \in \mathbb{Z}_N$. Tuple $(m, r_i, h_i, 0)$ is added to Hist[$\mathcal{H}$] and h_i is returned as the hash value of m, $\mathcal{H}(m) := h_i$.
  - If $i \in \mathcal{L}$ then h_i is defined as $h_i = {r_i}^e \cdot y \bmod N$ for a random integer $r_i \in \mathbb{Z}_N$. Tuple $(m, r_i, h_i, 1)$ is added to Hist[$\mathcal{H}$] and h_i is returned as the hash value of m, $\mathcal{H}(m) := h_i$.

Signing oracle Let m denote a chosen input message queried by $\mathcal{A}$.

- If $m \notin$ Hist[$\mathcal{H}$] then hash oracle $\mathcal{H}$ is invoked.
- Let (m, r_i, h_i, b_i) be the entry in Hist[$\mathcal{H}$] corresponding to input message m:
  - if $b_i = 1$ then signing oracle fails and stops;
  - if $b_i = 0$ then it returns $\sigma'_i = r_i$ as the signature on message m.

Reduction. Within time $\tau_{\mathcal{A}}$ and with probability $\epsilon_{\mathcal{A}}$, adversary $\mathcal{A}$ returns a forgery $\sigma^* = \mathcal{H}(m^*)^d \bmod N$ on a message m^*, after (at most) q_H hash queries and q_S signature queries, and m^* was not submitted to the signing oracle.

Without loss of generality, we assume that $\mathcal{A}$ has queried m^* to the random oracle, that is, that $m^* \in$ Hist[$\mathcal{H}$]. If $m^* = m_j$ for some $j \in \mathcal{L}$ then there exists a tuple $(m^*, r_j, h_j, 1) \in \mathcal{L}$ with $h_j = \mathcal{H}(m^*) = {r_j}^e \cdot y \bmod N$. Therefore, it follows that $\sigma^* \equiv ({r_j}^e\, y)^d \equiv r_j\, y^d \pmod{N}$. Hence, $x := \sigma^*/r_j \bmod N$ is a solution to the RSA challenge since $x^e \equiv y \pmod{N}$.

Complexity Analysis. The success probability ϵ of the reduction satisfies

$$\begin{aligned}
\epsilon &\geq \Pr[\text{perfect simulations}] \cdot \epsilon_{\mathcal{A}} \cdot \Pr[m^* = m_j \text{ for some } j \in \mathcal{L}] \\
&\geq P_t \cdot 1 \cdot \left(1 - \frac{\ell\, q_S}{q_S + q_H}\right) \cdot \epsilon_{\mathcal{A}} \cdot \frac{\ell}{q_S + q_H - q_S} \\
&= P_t \cdot \frac{\ell\,\left[(1-\ell)q_S + q_H\right]}{q_H(q_S + q_H)} \cdot \epsilon_A \qquad (\dagger)
\end{aligned}$$

by noting that the simulation of the random oracle is perfect and that m^* was not submitted to the signing oracle. The term P_t denotes the probability that transform exponent t —drawn as an odd integer in $[1, N)$— actually lies in $\mathcal{T}$.

Lemma 3.1. *For any integer $n \geq 2^{59}$, we have*

$$\varphi(n) > n/(2 \log\log n) \ .$$

Proof. Let $\gamma = 0.5772\ldots$ denote the Euler-Mascheroni constant. It is known that Euler's totient function satisfies the inequality

$$\varphi(n) > \frac{n}{e^\gamma \log\log n + \frac{3}{\log\log n}} \ .$$

Lemma follows by noting that for $n \geq 2^{59}$, $e^\gamma \log\log n + \frac{3}{\log\log n} < 2 \log\log n$. □

Since $\mathcal{T} \supset \left\{t \in \mathbb{Z} \mid 0 < t < \varphi(N) \text{ and } \gcd(t, \varphi(N)) = 1\right\}$, we therefore get

$$\begin{aligned}
P_t &= \frac{\#\,\mathcal{T}}{\#\text{ odd integers in } [1, N)} > \frac{\varphi\big(\varphi(N)\big)}{(N-1)/2} \\
&> \frac{\varphi(N)}{\log\log\varphi(N)} \cdot \frac{1}{N-1} > \frac{\varphi(N)}{\log\log N} \cdot \frac{1}{N} \\
&> \frac{1}{2\,(\log\log N)^2} \ .
\end{aligned}$$

Analogously to [11], we now determine the optimal value for ℓ. From Eq. (†), we obtain

$$(1-\ell)q_S + q_H + \ell(-q_S) = 0 \implies \ell = \frac{q_S + q_H}{2q_S} \ .$$

This yields

$$\epsilon \geq P_t \, \frac{q_S + q_H}{4q_S q_H} \, \epsilon_{\mathcal{A}} \geq \frac{P_t \cdot \epsilon_{\mathcal{A}}}{2q_S} > \frac{\epsilon_{\mathcal{A}}}{4q_S \, (\log\log N)^2} \ ,$$

assuming that $q_S \leq q_H$. The required time is $\tau \leq \tau_{\mathcal{A}} + (q_S + q_H)O(k^3)$ where k denotes the security parameter defining the scheme.

As an example, for a 3072-bit RSA modulus, the extra factor P_t, incurred in the security bound, amounts to less than 7 bits of security: $\frac{1}{2}(\log\log 2^{3072})^2 < 2^7$.

4 NOTES AND EXTENSIONS

4.1 Performance

The cost of our transformation is a single exponentiation. Since exponent t is public, there is no need to apply any special side-channel countermeasures. Any exponentiation method is acceptable.

4.2 Application to Additional RSA-Based Signature Schemes

The proposed construction readily extends to additional signature schemes based on the RSA assumption, such as RSA-PSS due to Bellare and Rogaway [4], and the variant of RSA-FDH due to Katz and Wang [15], which uses an additional random bit as an input to the hash function to achieve a tight proof of security. As we showed for RSA-FDH, there will be likewise an additional factor in the security bound, namely P_t, due to publication of t.

4.3 Application along with QVRSA

SGX uses QVRSA —a method that speeds up verification of RSA signatures using auxiliary values computed from the public key and the signature, which is explained in [8]. Our method does not interfere with the QVRSA verification scheme. It is completely oblivious to the transformation we perform in the backend in order to produce the signature, and we can nevertheless generate the auxiliary values $Q1$ and $Q2$ from the public key $\{N, e\}$.

4.4 Extension to RSA Encryption

The transformation in Sect. 2 extends to RSA encryption [3, 16]. In case we would like to decrypt ciphertexts corresponding to encryption under a public key unsupported by the HSM, we could generate a key-transform exponent in a similar way, and use it to transform one RSA ciphertext encrypted using the desired public key, to another, encrypted under a random public key corresponding to a private key stored in the HSM.

We note that contrary to proxy re-encryption, here we do not have the constraint that both "public" exponents are expected to be public. By removing e' from the adversarial view we are able to obtain extremely simple RSA-based re-encryption (we do not call it *proxy re-encryption*, to avoid confusion with a setting where both public keys are known).

5 CONCLUSIONS

We proposed a simple method that can be useful to work around limitations of certain cryptographic hardware modules. It enables obtaining RSA signatures verifiable using a given public exponent, when the cryptographic module does not support it. Using the RSA-FDH signature scheme as an example, we defined and proved security in the random-oracle model. We also discussed extensions and generalizations thereof.

ACKNOWLEDGMENTS

We thank David Wu from Stanford University for helpful feedback on an earlier version of this paper.

REFERENCES

[1] Giuseppe Ateniese and Susan Hohenberger. 2005. Proxy re-signatures: New definitions, algorithms, and applications. In *12th ACM Conference on Computer and Communications Security*, V. Atluri et al. (Eds.). ACM Press, 310–319.
[2] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, V. Ashby (Ed.). ACM Press, 62–73.
[3] Mihir Bellare and Phillip Rogaway. 1995. Optimal asymmetric encryption – How to encrypt with RSA. In *Advances in Cryptology – EUROCRYPT '94 (LNCS)*, A. De Santis (Ed.), Vol. 950. Springer, 92–111.
[4] Mihir Bellare and Phillip Rogaway. 1996. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96 (LNCS)*, U. M. Maurer (Ed.), Vol. 1070. Springer, 399–416.
[5] Matt Blaze, Gerrit Bleumer, and Martin Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology – EUROCRYPT '98 (LNCS)*, K. Nyberg (Ed.), Vol. 1403. Springer, 127–144.
[6] Dan Boneh. 1999. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS* 46, 2 (1999), 203–213. http://www.ams.org/notices/199902/boneh.pdf
[7] Dan Boneh, Glenn Durfee, and Yair Frankel. 1998. An attack on RSA given a small fraction of the private key bits. In *Advances in Cryptology – ASIACRYPT '98 (LNCS)*, K. Ohta and D. Pei (Eds.), Vol. 1514. Springer, 25–34.
[8] Dan Boneh and Shay Gueron. 2017. Surnaming schemes, fast verification, and applications to SGX technology. In *Topics in Cryptology – CT-RSA 2017 (LNCS)*, H. Handschuh (Ed.), Vol. 1514. Springer, 149–164.
[9] Don Coppersmith. 1997. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology* 10, 4 (1997), 233–260.
[10] Don Coppersmith, Matthew K. Franklin, Jacques Patarin, and Michael K. Reiter. 1996. Low-exponent RSA with related messages. In *Advances in Cryptology – EUROCRYPT '96 (LNCS)*, U. M. Maurer (Ed.), Vol. 1070. Springer, 1–9.
[11] Jean-Sébastien Coron. 2000. On the exact security of full domain hash. In *Advances in Cryptology – CRYPTO 2000 (LNCS)*, M. Bellare (Ed.), Vol. 1880. Springer, 229–235.
[12] Johan Håstad. 1988. Solving simultaneous modular equations of low degree. *SIAM J. Comput.* 17, 2 (1988), 336–341.
[13] Intel Corporation [n. d.]. Intel Software Guard Extensions. https://software.intel.com/en-us/sgx.
[14] Intel Corporation 2018. *Signing and whitelisting for Intel Software Guard Extensions (Intel SGX) enclaves.* White Paper.
[15] Jonathan Katz and Nan Wang. 2003. Efficiency improvements for signature schemes with tight security reductions. In *10th ACM Conference on Computer and Communications Security*, S. Jajodia et al. (Eds.). ACM Press, 155–164.
[16] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.
[17] Hideo Shimizu. 1996. On the improvement of the Håstad bound. In *1996 IEICE Fall Conference*, Vol. A-162. (In Japanese).
[18] Michael J. Wiener. 1990. Cryptanalysis of short RSA secret exponents. *IEEE Trans. Inf. Theory* 36, 3 (1990), 553–558.