Server-Supported RSA Signatures for Mobile Devices

Ahto Buldas[1,2](✉), Aivo Kalu[1], Peeter Laud[1], and Mart Oruaas[1]

[1] Cybernetica AS, Tallinn, Estonia
ahto.buldas@cyber.ee
[2] Tallinn University of Technology, Tallinn, Estonia

Abstract. We propose a new method for shared RSA signing between the user and the server so that: (a) the server alone is unable to create valid signatures; (b) having the client's share, it is not possible to create a signature without the server; (c) the server detects cloned client's shares and blocks the service; (d) having the password-encrypted client's share, the dictionary attacks cannot be performed without alerting the server; (e) the composite RSA signature "looks like" an ordinary RSA signature and verifies with standard crypto-libraries. We use a modification of the four-prime RSA scheme of Damgård, Mikkelsen and Skeltved from 2015, where the client and the server have independent RSA private keys. As their scheme is vulnerable to dictionary attacks, in our scheme, the client's RSA private exponent is additively shared between server and client. Our scheme has been deployed and has over 200,000 users.

1 Introduction

Digital signature mechanisms require secure storage of private keys. It is often recommended to hold keys in special hardware (like smart-cards). This is an expensive solution for wide employment of digital signatures (e.g. national digital ID). Moreover, nowadays people use mobile devices for their everyday business, as well as for communicating with e-government services. Mobile devices may not have a possibility to physically connect to a card-reader, the connection interfaces change rapidly, and after all, connecting a card-reader with a mobile phone would just be inconvenient. The cryptographic algorithms used for digital signatures may become insecure and the key size insufficient. Changing the algorithm or the key size would generally mean physical replacement of all smart-cards in use.

Software is much easier to change. Mobile devices update their software automatically so that the users often do not even notice the updating process. From economical perspective, digital signature solutions based solely on software are extremely appealing. The hardest thing to solve in software-based digital signature schemes is private key management. Keys stored in the static memory of a mobile device or any other type of computer can easily be cloned by attackers who gain access to the memory. With cloned keys, attackers can create unlimited amounts of forged signatures that are indistinguishable from the genuine ones.

This work has been supported by Estonian Research Council, grant No. IUT27-1.

S.N. Foley et al. (Eds.): ESORICS 2017, Part I, LNCS 10492, pp. 315–333, 2017.
DOI: 10.1007/978-3-319-66402-6_19

Private keys may be stored in encrypted form, where the decryption key is derived from a password entered by the owner of the device. But practice shows that human-memorisable passwords do not withstand dictionary attacks.

One way to make software-based digital signatures more secure is to share the signature key between the mobile device and a service, so that a correct signature can only be created when the mobile device and the service cooperate. So even if the user's share of the signature key is cloned, the use of the clone requires communication with the service. If the user's key-share is chosen randomly and is camouflaged (i.e. encrypted with the user's password in a proper way [18,19,21]), an off-line dictionary attack will not be possible, because the attacker (without communicating with the server) cannot distinguish the right password from the wrong guesses. If the attacker uses the service for the dictionary attack, such an attempt is recognisable for the service, and after a fixed number wrong guesses, the service may block the client and refuse to cooperate.

Shared keys are easy to generate for one party. In this case, the user's mobile device generates both shares, keeps one share, sends the second share to the service, and deletes the second share. Such a solution however does not protect against an attack where the mobile device is under adversary's control during key generation. After such an attack, the attacker is again able to create unlimited number of forged signatures without communicating with the server. If the server is the party who generates the shares and gives the share to the client, such an attack is not possible but in that case, the service (if abused by insiders) would be able to forge the signatures in unlimited way.

Hence, to withstand dictionary attacks and at the same time to avoid the abuse of the key by potentially malicious servers, the mobile device and the server must generate their shares in such a way that none of the parties at any time has the complete private key of the user. Such cryptographic protocols exist but have drawbacks. The general multi-party computation methods are complex and inefficient. Some methods assume third parties' involvement during key generation, such as trusted dealers.

Recently, Damgård, Mikkelsen and Skeltved [11] proposed an elegant scheme in which two parties can generate their RSA key shares completely independently and with the same computational effort than generating ordinary RSA keys. Their scheme has a drawback though. If one wants to implement their scheme as a software application for a mobile device, it turns out that even if the private key is perfectly camouflaged (password-encrypted), the attacker always has a reference point for a dictionary attack. This is because the client's public modulus is needed to create the client's share of the signature and hence either the public modulus is stored in the device in open form or is recoverable via password-based decryption. As a reference point, an attacker may check the relationship between the decrypted private exponent and the public modulus.

We present the *Smart-ID* scheme, a modification of the scheme of [11] to make it invulnerable to dictionary attacks. The main idea is to additively share the client's RSA private exponent so that the camouflaged part of the key is completely random and gives no reference points for dictionary attacks.

We also consider some ways of making the scheme even stronger by adding a mechanism that enables the service to discover clones of the client's key and to block the service timely.

2 State of the Art

RSA: The RSA signature scheme [22] is one of the most widely used digital signatures. A message $m \in \mathbb{Z}_n = \{0, \ldots, n-1\}$ is signed with a modular power function $\sigma(m) = m^d \mod n$, where d is the secret exponent and $n = pq$ is a product of two large prime numbers p and q. The verification check of a signed message (m, σ) also involves a power function $\nu(\sigma) = \sigma^e \bmod n$, where e is the public exponent. A signed message (m, σ) verifies correctly, if $\nu(\sigma) = m$. The public and the private exponents satisfy $ed \equiv 1 \pmod{\varphi(n)}$, where $\varphi(n) = (p-1)(q-1)$ is the Euler's totient function.

Shared RSA: Suppose the private exponent d is the sum of two random components $d \equiv d' + d'' \pmod{\varphi(n)}$, where d' and d'' are held by two separate parties (say, Client and Server). To sign a message m cooperatively, the parties create their signature shares $\sigma' = m^{d'} \bmod n$ and $\sigma'' = m^{d''} \bmod n$. The shares are then combined by

$$\sigma = \sigma' \cdot \sigma'' \bmod n = m^{d'} \cdot m^{d''} \bmod n = m^{d'+d''} \bmod n = m^d \mod n \ ,$$

which is the ordinary RSA signature σ of m. This is called *additive sharing* of RSA signature. Only the two parties together can create verifiable signatures.

The idea of shared key approach was first presented by Desmedt and Fraenkel [12,13]. For the RSA signature scheme [22], the shared keys approach is studied in [8,15,23] and the mobile device and server case in [2,7,9,21] but these works do not investigate the problem of generating keys in a distributed way. It is assumed that shares of the key are generated by a trusted dealer.

Shared Generation of RSA Keys: Distributed generation of shared RSA keys has also been thoroughly studied. The first practically implementable solution was proposed by Boneh and Franklin [4,5]. The following schemes [6,10,14,16,17] are just variations of the original scheme [4]. The main idea is to generate a candidate RSA modulus $n = pq$ (where p, q are just random numbers) using multi-party computation, so that p and q will be additively shared between the parties. A special bi-primality test is then applied to n. The candidate n can be used if both p and q are prime. Hence, the average number of attempts is quadratic in the size of n, which means that the key generation time is very large – hundreds of times slower than the original RSA key generation.

Damgård-Mikkelsen-Skeltved Scheme: An elegant and efficient solution to the problem of shared generation was proposed in [11]. In their scheme, after fixing the public exponent e, the two parties first locally generate their own RSA public keys (n_1, e) and (n_2, e) and the corresponding private exponents d_1 and d_2. The final (composite) public key is $(n_1 n_2, e)$. To sign a message

$m \in \mathbb{Z}_{n_1 n_2}$, both parties first create their own signatures $\sigma_1 = m^{d_1} \mod n_1$ and $\sigma_2 = m^{d_2} \mod n_2$. They will then use the Chinese Remainder Theorem (CRT) to compute the final signature $\sigma = C_{n_1,n_2}(\sigma_1, \sigma_2) \in \mathbb{Z}_{n_1 n_2}$, satisfying $\sigma \equiv \sigma_i \pmod{n_i}$ for $i \in \{1, 2\}$. To verify such signature, one simply checks whether $\sigma^e \equiv m \pmod{n_1 n_2}$. Hence any existing software supporting RSA signatures is able to verify the signatures of [11] without modifications.

The problem with this solution is that dictionary attacks are still possible, even if the client's private exponent is encrypted – the client's public modulus (say n_1) is public and can be used to verify the guessed passwords.

Camenisch et al. Scheme: A server-assisted RSA signature scheme was proposed in [7]. In their scheme, the client is authenticated with the help of password, designed so that the dictionary attacks against it are impossible. While aiming for a range of advanced properties, such as privacy against the server, and universally composable security, the signing key in their scheme is generated fully by the client, and then shared with the server. Therefore, an adversary who reads the client device during key generation has a power to create valid signatures without contacting the server.

Dictionary Attacks: An adversary, having a dictionary of passwords, tries them one by one until the right one has been found. For such attack:

– The number of possible passwords has to be relatively small. Random cryptographic keys with a lot of entropy (≥ 80 bits) cannot be successfully guessed.
– It must be possible to recognize the right password [3, Definition 3.10].

If the private exponent d of the RSA key is encrypted with a password pwd, and the adversary has both the ciphertext c_d and the public key (n, e), then it can verify its guess pwd^* by generating a random $m \in \mathbb{Z}_n$ and checking whether

$$(m^e)^{\mathsf{dec}_{pwd^*}(c_d)} \equiv m \pmod{n} .$$

This is the case for the scheme in [11], where the client's device must contain c_d and the client's modulus n_1, and the public exponent e is known to everybody.

Shared RSA may be used to take away the point of reference that the adversary uses to check the correctness of its guesses. This has been done in [7], but their scheme has other undesirable properties, as described previously.

3 New Scheme

We will now describe our scheme with the properties listed in the abstract. None of the previously proposed schemes have all these properties. The main idea of our solution is that we use a scheme similar to [11] where client and server have independent RSA keys. We make their scheme resistant to dictionary attacks. To simplify the presentation, we assume that each client of the scheme has only a single key. Then we can talk about either blocking a key, or blocking a client. We start with a definition.

Definition 1. *A prime number p is an (ℓ, s)-*safe prime*, if $p = 2ap'_1 \cdots p'_k + 1$, where $p'_i > s$ are prime numbers, and $1 \leq a \leq \ell$.*

3.1 Description of the Scheme

Setup: Let the desired security level of the scheme be η bits. From η, suitable values for ℓ and s, as well as the RSA modulus length k are selected. An example of such selection is given in Sect. 6. The numbers T_0 of wrong password guesses for a client, and the public exponent e (e.g. 3, or $2^{2^4}+1$) are also fixed.

For each active client C, the server stores the values n_{1C}, n_{2C}, d''_{1C}, d_{2C}, r_C, T_C. In following, we drop the subscript C if it is clear from the context. Here n_1 and n_2 are k-bit RSA moduli, $d''_1 \in \mathbb{Z}_{n_1}$, $d_2 \in \mathbb{Z}_{n_2}$, r is an η-bit string, and T is the wrong password counter for the given client. In this scheme, the quantities n_1, n_2, d_2 are the same as in [11]. The client's private exponent d_1 is additively shared as $d_1 = d'_1 + d''_1 \pmod{\varphi(n_1)}$ between the client and the server. The one-time password r is used to detect clones of client's signing functionality.

Let $\mathcal{P} \subseteq \{0,1\}^l$ be the set of possible passwords. Given a password $pwd \in \mathcal{P}$, there has to be a client-specific process of turning pwd into a value $d'_1 \in \mathbb{Z}_{n_1}$. Given a generic black-box pseudo-random function $\Phi : \{0,1\}^{l+8} \rightarrow \{0,1\}^k$, a possible way to construct such d'_1 is given in Algorithm 1. Different clients use different functions Φ. In practice, $\Phi(\cdot)$ is replaced by a pseudo-random function $F(u,\cdot)$, where u is a sufficiently long random bit-string. Such F can be constructed from a block cipher.

Algorithm 1. $\texttt{genShare}^{\Phi}(pwd, n_1)$- client's key share generation using a generic blck-box PRF Φ

```
for  $s \in \{0,1,2,\ldots,255\}$ do
    $d'_1 \leftarrow \Phi(pwd\|s)$;
    if $d'_1 < n_1$ then
        return $d'_1$;
return $\perp$;
```

We see that Algorithm 1 may fail, but its probability of failure is less that 2^{-256}. Indeed, the probability of a single iteration failing is less than 1/2, because $n_1/2^{k-1} \geq 1/2$. If Φ is a random function, then all $\Phi(pwd\|0), \ldots, \Phi(pwd\|255)$ are independent, hence we can multiply the probabilities.

To sign a message M, a cryptographic hash $H(M)$ is computed and a padding P is added. The hashed and padded message $m = P(H(M))$ is then input to the signing protocol. The setup of the scheme includes fixing H and P [25].

Key generation: The client C finds two (ℓ,s)-safe primes p_1, q_1 with gcd(p_1-1, e)=gcd(q_1-1, e)=1, computes $n_1 = p_1 q_1$ and $d_1 = e^{-1} \pmod{\varphi(n_1)}$, and stores n_1.

The client gets a password $pwd \in \mathcal{P}$ from the user, generates and stores a random bit-string u, computes $d'_1 = \texttt{genShare}^{F(u,\cdot)}(pwd, n_1)$ and $d''_1 = d_1 - d'_1 \pmod{\varphi(n_1)}$, generates and stores a random bit-string r. It sends $\langle d''_1, n_1, r\rangle$ to the server. All communication between C and S takes place over secure channels.

The server S generates two (ℓ, s)-safe primes p_2, q_2 satisfying $\gcd(p_2-1, e) = \gcd(q_2-1, e) = 1$. It computes $n_2 = p_2 q_2$, $n = n_1 n_2$, and $d_2 = e^{-1} \pmod{\varphi(n_2)}$. It takes $T = T_0$ and stores $\langle n_1, n_2, d_1'', d_2, r, T\rangle$. It sends n back to the client. The public key of C is (n, e). The client securely deletes all values except $\langle n, n_1, u, r\rangle$.

Signing: To sign a (hashed and padded) message m, the client C gets a password *pwd* from the user, finds $d_1' = \mathtt{genShare}^{F(u,\cdot)}(pwd, n_1)$ and the *signature share* $y = m^{d_1'} \pmod{n_1}$, picks a random $r' \leftarrow \{0,1\}^{\eta}$ and sends $\langle y, m, r, r'\rangle$ to S.

The server S checks that C is active, looks up its record $\langle n_1, n_2, d_1'', d_2, r, T\rangle$, computes the *client's signature* $s_1 = y \cdot m^{d_1''} \pmod{n_1}$ and checks its correctness by verifying if $s_1^e = m \pmod{n_1}$. If not, S decrements T and drops the request. If $T = 0$, the server deactivates the client.

If the signature check succeeds, S checks if r in the request coincides with server's copy of r. In case of match, S computes $s_2 = m^{d_2} \mod n_2$, creates the composite signature $s = C_{n_1,n_2}(s_1, s_2)$, sends the signature reply $\langle s, m\rangle$ back to the client's device, stores the new password r' as r (expecting that the next signature request will contain r'), and assigns T_0 to T. If the signature check succeeds but r in the request differs from the stored value, the server deactivates the client. Having received back the signature, C replaces its stored r with r'.

Verification: To verify the signature σ for a hashed and padded message m, with respect to a public key $(n_1 n_2, e)$, one uses the standard RSA verification scheme by just checking that $m^e = m \pmod{n_1 n_2}$.

3.2 Employed Detection Mechanisms

Key Clone Detection: For detection of fraud, the signing protocol exchanges additional information between the server and the client and after every new signature, a common (to client and server) random one-time password is formed. The one-time password that was formed during the previous signature creation is a part of the next signature request and is verified by the server during every signing operation. If the state-vector verification fails but the signature request itself verifies correctly (i.e. the partial signature is authentic), the server knows that there are two copies of the client's private key in use (this is the most likely cause), and deactivates the client immediately. The clone detection mechanism can be added as an additional protection layer to any two party (client-server) type of a signature scheme, assuming that in the signing protocol, the client's share s_1 of the signature can be verified by the server during the protocol.

If d_1' has been cloned, the adversary becomes able to impersonate the client. The main idea of the method is that the client must know the content of the previously made queries, and this knowledge is verified by the server during every signing request. If there are two identical copies of the client private key owned by two different parties, then only one of these parties will be able to continue using the service: namely, the party who first makes the next signing request. This is because if then also the second party will make a request, it has no knowledge of the other party's request and will not be served.

The server will deactivate the client, once it has received a request with correctly verifying client's signature share, but with incorrect previous query identifier. Such a query is a strong evidence of the existence of two copies of client's private key.

Periodic Dummy Requests: For faster detection of key abuse, the device may send periodic dummy signature requests, which are exactly the same as real signing requests. They require authentication at the server side, but do not create new signatures. The server has to reply with a dummy reply that is processed at the device side in the same way as ordinary requests, except that the device knows that the reply does not require any processing. The time between two dummy requests is the maximal time the adversary who has a cloned share of the client's key (or the clone of the whole key) is able to create forgeries.

4 Robust Implementation

There are the following general types of attacks against a client's signing device:

- **Device Read**: The adversary has a short-time access to the passive memory-content of the device, like the encrypted key file via a cloud-stored backup. The encrypted key file can then be a subject to dictionary attacks.
- **Device Memory Read**: The adversary obtains a copy of the active memory of the device, which may contain the client's private key share.
- **Device Memory Read During Key Generation**: The adversary reads active memory of the device during key generation and obtains client's private exponent (not just the client's share).
- **Device Malware**: The adversary inserts an active trojan to the signature device that could stay in the device for arbitrarily long time, i.e. until it is detected or is removed on a command of the adversary.
- **Server Internal Attack**: The adversary obtains client-specific secrets that the service has, or even gets access to server's private key. Insider attacks fall into this category.

We analyze the vulnerabilities of possible implementations (represented as a combination of features) of a server-supported personal signature solution based on the new shared RSA signature scheme. We consider the following features:

- **Independent Key Generation**: This means that the server and the client generate their keys completely independently. This is the key feature of the scheme of [11].
- **Client's Key is Shared with Server**: This means that the client's private key is shared between the device and the server.
- **Clone Detection**: This means that the a special protocol is used for key clone detection, which blocks the service once both copies of the key are used at least once (after cloning). If t_u is time until the next usage of the device, then the adversary who cloned the key has t_u units of time available to abuse the cloned key. After t_u, in case the genuine device also exists, it sends the

next signature request to the server and the service is blocked. If the client issues periodic dummy requests to the server as described in Sect. 3.2, then t_u has a well-defined upper bound.

There are 6 meaningful combinations of these features. We analyse their vulnerability and also compare them to the solution where client's private key is held in a smart-card. We assume that the smart-card itself generates the client's key and is tamper-proof. The combined solutions are denoted as follows:

– **4RSA**: The original 4RSA proposed by Damgård et al. [11]
– **S**: An ordinary (additively) shared RSA scheme
– **4D**: 4RSA complemented with the clone detection mechanism
– **SD**: S complemented with the clone detection mechanism
– **S4**: 4RSA where the client's private exponent is shared
– **S4D**: The solution that combines 4D and S4

Table 1. Comparison of vulnerabilities of the implementations: -means invulnerable, $+t$ means limited t-time vulnerability, +means unlimited vulnerability.

Name	Indep. key gen.	Client's key shared	Clone detection	Service inner	Device read	Device mem. read	Mem. read during key gen	Device malware
Smart-card	no	no	no	-	-	-	-	+
S4D	yes	yes	yes	-	-	$+t_u$	$+t_u$	+
SD	no	yes	yes	-	-	$+t_u$	+	+
S4	yes	yes	no	-	-	+	+	+
S	no	yes	no	-	-	+	+	+
4D	yes	no	yes	-	$+t_u$	$+t_u$	$+t_u$	+
4RSA	yes	no	no	-	+	+	+	+

The comparison of vulnerabilities of these solutions are summarised in Table 1. All solutions are vulnerable against malware attacks because active malware is able to change the hash value that is intended to be signed and thereby to forge any signature. None of the solutions is vulnerable to inner attacks against the service because due to the security of RSA, the server is not able to deduce useful information about client's private key, having only the public parameters, and the data disclosed to the server.

4.1 Server's Key: Client-Specific or Common?

Should the server have just one private key or should the private key be client-specific? It turns out that that in the case of common server key, the solution S4D presented in Sect. 3 has unlimited vulnerability against the Memory Read attacks during key generation. If the adversary is one of the client's, say A,

Algorithm 2. Existential forgery via adaptive chosen message attack

```
$(p_1, q_1, d_1) \leftarrow \texttt{genKey}(k, e)$;
$n_1 \leftarrow p_1 \cdot q_1$;
$(M, \sigma) \leftarrow A^{H,\Sigma}(n_1)$;
if $\sigma \equiv P(H(M))^{d_1} \pmod{n_1}$ and $A$ never called $\Sigma(H(M))$ then
    return 1;
else
    return 0;
```

who has cloned a private key of another client B, then A can forge B's signature on m as follows. First, it signs m herself by sending a signing request to the server. Server sends back the composite signature $C_{n_A,n_S}(\sigma_A(m), \sigma_S(m))$. After that, A uses the stolen copy of B's key to create $\sigma_B(m)$ and forms the composite signature $C_{n_B,n_S}(\sigma_B(m), \sigma_S(m))$. Note that the clone detection mechanism will not activate, because there is no communication that involved the cloned key. Hence, the server's key has to be client-specific.

5 Proofs of Security

The notion of exact security (first proposed in [1]) is needed when drawing practical conclusions on security proofs. We use the definition from [20]:

Definition 2. *A cryptographic scheme is S-secure if any t-time adversary has success $\delta \leq \frac{t}{S}$, i.e. if every adversary has time-success ratio $\frac{t}{\delta} \geq S$.*

For real-life cryptography, the notion of security bits is often used. For example, the statement that RSA with 2048-bit modulus has 112 bits of security [24] means, that the running time of the adversary is measured in time units equal to the time of encrypting one single block with a typical block-cipher (like AES).

Definition 3. *A cryptographic scheme has k bits of security, if any adversary with running time of T block-cipher units has success $\delta \leq T/2^k$.*

In security proofs, we assume that (for certain S) the RSA signature $\Sigma = P(\cdot)^{d_1}$ mod n_1 together with the padding scheme P is S-secure against existential forgeries via adaptive chosen message attacks, where H is a hash function which we model as a random oracle. Such attacks are defined as follows.

Definition 4. *In an adaptive chosen message attack, an adversary $A^{H,\Sigma}(n_1)$ having access to the signing oracle produces a correct message-signature pair $M, \Sigma(H(M))$, without querying Σ with $H(M)$ (Algorithm 2).*

For storing client's share securely, we need pseudo-random functions.

Definition 5. *By an S-secure* Pseudo-Random Function *we mean an efficiently computable two-argument function $F\colon \{0,1\}^n \times \{0,1\}^p \to \{0,1\}^m$, such that if the first argument u is randomly chosen then the one-argument function $F(u,\cdot)$ (given to the distinguisher as a black box without direct access to u) is S-indistinguishable from the truly random function $\mathcal{F}$ of the same type, i.e.*

$$\left| \operatorname*{Prob}_{u}\left[1 \leftarrow D^{F(u,\cdot)}\right] - \operatorname*{Prob}_{\mathcal{F}}\left[1 \leftarrow D^{\mathcal{F}}\right] \right| \le \frac{t}{S} \ ,$$

for any t-time distinguisher D, where $u \leftarrow \{0,1\}^n$ and $\mathcal{F}$ is a function chosen randomly and uniformly from the set of all functions of type $\{0,1\}^p \to \{0,1\}^m$.

Outline of Proofs: We will prove the following aspects of security:

- **Security of the composition procedure**: The composed signature scheme is almost as secure as the underlying RSA scheme.
- **Security against malicious servers**: Having the public key of the client and the server-share of client's private key, and being able to use client and a signing-oracle, the adversary is unable to sign a message that has not been used as an oracle query.
- **Security against device read**: Having the public key of the client and the password-encrypted private key share, the adversary is not able to create a forged signature with probability much larger than $\frac{T}{K}$, where K is the total number of passwords (PINs), assuming that the password is chosen uniformly from the set of all possible passwords, and T is the maximum number of consecutive faulty trials.
- **Security against device memory read**: Having the public key of the client, the actual private key share, and the one-time password r, the adversary can create forged signatures only until the legitimate client makes the next signing request.

Security proofs depend on the type of primes. Some types of primes may offer better attack-resistance, while other types of primes might be easier to generate.

5.1 Security of the Composition Procedure

We show that if an attacker succeeds in adaptive chosen message attack against the composite signature, then there is an attacker that succeeds in adaptive chosen message attack against the ordinary RSA signature. Let Σ be an oracle that, given as input a hashed message m, outputs the composite signature

$$\sigma(P(m)) = C_{n_1,n_2}(\sigma_1(P(m)), \sigma_2(P(m))) \ .$$

Let Σ' be an oracle such that $\Sigma'(m) = \sigma_2(P(m))$.

Theorem 1. *If RSA is S-secure against existential forgeries via adaptive chosen message attack, then the composite signature is about $\frac{S}{t_{\text{ex}}}$-secure against the same attack, where t_{ex} is the time for one modular exponentiation.*

Proof. Let $(m, \sigma(P(m))) \leftarrow A^{\Sigma}(n_1 n_2)$ be a t-time adversary that, with probability δ, produces a valid signature for a message m that was never queried via the Σ-oracle. We construct an adversary $(m, \sigma_2(P(m))) \leftarrow A_2^{\Sigma'}(n_2, e)$ that creates a valid signature of a message m that was never queried via the Σ'-oracle. The adversary A_2 generates an RSA key with public modulus n_1 and with secret exponent d_1 such that $ed_1 \equiv 1 \pmod{\varphi(n_1)}$ and then simulates $(m, \sigma(P(m))) \leftarrow A^{\Sigma}(n_1 n_2)$ so that the $\Sigma(m)$-calls are simulated by calling $\sigma_2 \leftarrow \Sigma_2(m)$, computing $\sigma_1 \leftarrow P(m)^{d_1} \mod n_1$, and finally combining σ_1 and σ_2 to the composite signature $\sigma \leftarrow C_{n_1,n_2}(\sigma_1, \sigma_2)$. If A produces a valid signature $(m, \sigma(P(m)))$, then A_2 decomposes $\sigma(P(m))$ to σ_1 and σ_2, and outputs (m, σ_2). If A did not make the oracle call $\Sigma(m)$, then A_2 did not call Σ' with m. Hence, A_2 (like A) succeeds with probability δ. The running time of A_2 does not exceed $t_{\text{gen}} + tt_{\text{ex}}$, where t_{gen} is the time for RSA key generation. Hence, as σ_2 is S-secure,

$$\delta \leq \frac{t_{\text{gen}} + tt_{\text{ex}}}{S} \leq \frac{t(t_{\text{ex}} + \frac{t_{\text{gen}}}{t})}{S} \approx \frac{tt_{\text{ex}}}{S} ,$$

assuming that $t \gg t_{\text{gen}}$ which means that σ is about $\frac{S}{t_{\text{ex}}}$-secure. □

5.2 Security Against Malicious Servers

We consider a malicious server as an adversary A that has a share d_1'' of the client's private modulus d_1 and also has a connection to client's signature device that sends signing requests to the server. We assume that A is able to use such a connection as an oracle $\Sigma_{d_1'}$, i.e. to choose messages m, send m to the oracle and obtain $\Sigma_{d_1'}(m) = P(m)^{d_1'} \mod n$. Though, in practice, the server cannot choose the message m to be signed, we may assume that it does. The goal of $A^{H,\Sigma_{d_1'}}(d_1'')$ is to produce a message M and the signature $P(H(M))^d \mod n$ such that the query $\Sigma_{d_1'}(H(M))$ was never made. Algorithm 3 describes such an attacking scenario. In the real scheme, Φ is $F(u, \cdot)$ that is assumed to be a PRF. In the *idealized scheme*, Φ is a truly random function.

Algorithm 3. Existential forgery by malicious server

```
$(p_1, q_1, d_1) \leftarrow \texttt{genKey}(k, e)$;
$n_1 \leftarrow p_1 \cdot q_1$;
$p \leftarrow \mathcal{P}$;
$d_1' \leftarrow \texttt{genShare}^{\Phi}(p, n_1)$;
$d_1'' \leftarrow d_1 - d_1' \mod \varphi(n_1)$;
$(M, \sigma) \leftarrow A^{H,\Sigma_{d_1'}}(n_1, d_1'')$;
if $\sigma \equiv P(H(M))^{d_1} \pmod{n_1}$ and $A$ never called $\Sigma_{d_1'}(H(M))$ then
    return 1;
else
    return 0;
```

Theorem 2. *If RSA is S-secure against existential forgeries via adaptive chosen message attack and $F(u,\cdot)$ is an S-secure PRF, then the shared signature system is $\frac{S}{2t_{\mathrm{ex}}}$-secure against malicious servers, where t_{ex} denotes the time needed for one modular exponentiation.*

Proof. Let $A^{H,\Sigma_{d'_1}}$ be a t-time adversary that with probability δ produces a pair M, $\Sigma(H(M))$ without calling $\Sigma_{d'_1}$ with $H(M)$. If instead, we had the flipped version of the idealized scheme where the parts d'_1 and d''_1 are exchanged and the server's share is just a uniformly distributed random number $r \leftarrow \mathbb{Z}_{n_1}$, by Lemmas 1 and 2, the success of $A^{H,\Sigma_{d'_1}}$ is at least $\delta - \frac{t+t_{\mathrm{gen}}+t_{\mathrm{ex}}}{S} - \frac{2}{p} - \frac{2}{q}$.

We construct an adversary $\underline{A}^{H,\Sigma}$ with running time $t' \approx t$ that succeeds in the original adaptive chosen message attack (Algorithm 2) with probability $\delta - \frac{t+t_{\mathrm{gen}}+t_{\mathrm{ex}}}{S} - \frac{2}{p} - \frac{2}{q}$ (against the flipped idealized scheme). The adversary $\underline{A}^{H,\Sigma}(n_1)$ first picks $r' \leftarrow \mathbb{Z}_{n_1}$ at random and then simulates $A^{H,\Sigma_{d'_1}}$, so that the calls $\Sigma_{d'_1}(m)$ are answered with $\Sigma(m)\cdot P(m)^{-r'} \mod n_1$. As the simulation is perfect, the success of $\underline{A}$ is $\delta - \frac{t+t_{\mathrm{gen}}+t_{\mathrm{ex}}}{S} - \frac{2}{p} - \frac{2}{q}$. The running time of $\underline{A}$ does not exceed $t + t(t_{\mathrm{mul}} + t_{\mathrm{ex}})$. Thus, $\delta \le \frac{t(1+t_{mul}+t_{exp})}{S} + \frac{t+t_{\mathrm{gen}}+t_{\mathrm{ex}}}{S} + \frac{2}{p} + \frac{2}{q}$. Assuming $t \ge t_{\mathrm{ex}}$ and $t_{\mathrm{ex}} \ge t_{\mathrm{mul}} + 4 + \frac{2S}{p} + \frac{2S}{q}$ we have $\frac{t}{\delta} \ge \frac{S}{4+t_{\mathrm{mul}}+t_{\mathrm{ex}}+\frac{2S}{p}+\frac{2S}{q}} \ge \frac{S}{2t_{\mathrm{ex}}}$. □

Lemma 1. *If $F(u,\cdot)$ is an S'-secure PRF, any t-time adversary A that succeeds in the malicious server attack against the real scheme with probability δ, succeeds against the idealized scheme with probability at least $\delta - \frac{t+t_{\mathrm{gen}}+t_{\mathrm{ex}}}{S'}$.*

Proof. Otherwise, the $(t+t_{\mathrm{gen}}+t_{\mathrm{ex}})$-time distinguisher D^{ϕ} defined by Algorithm 3 would have success $\delta' > \frac{t+t_{\mathrm{gen}}+t_{\mathrm{ex}}}{S'}$, contradicting the S'-security of $F(u,\cdot)$. □

Lemma 2. *If in the idealized scheme, d'_1 and d''_1 are the client's and the server part of the client's private modulus d_1 then the distributions of (d'_1, d''_1) and (d''_1, d'_1) are statistically $\left(\frac{2}{p}+\frac{2}{q}-\frac{2}{pq}\right)$-indistinguishable, which means that flipping the components d'_1 and d''_1 can change the success probability of any adversary (regardless of the definition of the success) only by $\frac{2}{p}+\frac{2}{q}-\frac{2}{pq}$.*

Proof. Consider an attacking scenario that involves our signature scheme and an adversary A. We construct a distinguisher $D(x,y)$ which simulates the attacking scenario, except instead of generating the parts d'_1, d''_2 in the proper way, D just assigns $d'_1 \leftarrow x$ and $d''_2 \leftarrow y$. The distinguisher outputs 1 if and only if A succeeds in the simulation. Due to the statistical closeness of uniform distributions over $\mathbb{Z}_{pq}$ and $\mathbb{Z}_{\varphi(pq)}$, and the involutory nature of constructing d''_1 from d'_1, the success of D cannot exceed $\frac{2}{p}+\frac{2}{q}-\frac{2}{pq}$. As the success of the distinguisher is by definition the difference between A's success in the original scheme and $A's$ success in the flipped version of the scheme, this difference does not exceed $\frac{2}{p}+\frac{2}{q}-\frac{2}{pq}$. □

5.3 Security Against Device Read

The adversary has obtained the random value u stored in the device in open form. This u is combined with user's password p to obtain the client's share d'_1.

Adversary's access to the server is modelled as an oracle S with internal state. It receives queries of the form $(m, m^{d_1'} \bmod n_1)$ and returns $m^{d_1} \bmod n_1$ if the query is in such form. Otherwise, S returns $\perp$. After T_0 consecutive $\perp$-returns, S "blocks" and will return only $\perp$ even if the queries were correctly formed.

Algorithm 4. Existential forgery via device read

```
$(p_1, q_1, d_1) \leftarrow \texttt{genKey}(k, e)$;
$n_1 \leftarrow p_1 \cdot q_1$;
$p \leftarrow \mathcal{P}$;
$u \leftarrow \{0,1\}^m$;
$d_1' \leftarrow \texttt{genShare}^{F(u,\cdot)}(p, n_1)$;
$d_1'' \leftarrow d_1 - d_1' \bmod \varphi(n_1)$;
$(M, \sigma) \leftarrow A^{H,\Sigma,\mathsf{S}}(n_1, u)$;
if $\sigma \equiv P(H(M))^{d_1} \pmod{n_1}$ and $A$ never called $\Sigma(H(M))$ then
    return 1;
else
    return 0;
```

Definition 6. *For any two primes p, q, a padding function $P\colon \{0,1\}^h \to \mathbb{Z}_{pq}$, a positive integer s, and a uniformly random $m \leftarrow \mathbb{Z}_{pq}$, we use the notation*

$$\pi^P_{p,q}(s) = \underset{m}{Prob}\left[\operatorname{ord}(P(m)) < s\right] \quad .$$

Theorem 3. *If RSA signatures are S-secure against adaptive chosen message attack and $F(u,\cdot)$ is S'-secure PRF, then for every s, any t-time adversary $A^{H,\Sigma,\mathsf{S}}$ succeeds in existential forgery (Algorithm 4) with probability*

$$\delta \leq \frac{T_0}{K} + t \cdot \frac{K^2}{2s} + t \cdot \frac{K}{S'}(t_{\mathsf{ex}} + \log_2 K) + t \cdot \pi^P_{p,q}(s) + \frac{t^2}{2^h} + \frac{t t_{\mathrm{ex}}}{S} \quad ,$$

where K is the number of possible passwords (PINs) and T_0 is the maximum allowed consecutive false password trials.

Proof. Let $A^{H,\Sigma,\mathsf{S}}(n_1, u)$ be a t-time adversary that succeeds in the existential forgery attack via device read with probability δ. We may assume without loss of generality that $A^{H,\Sigma,\mathsf{S}}(n_1, u)$ never repeats any oracle calls (with the same input), and once it outputs (M, σ), it has made a call $m \leftarrow H(M)$. We construct an adversary $\underline{A}^{H,\Sigma}(n_1)$ that simulates $A^{H,\Sigma,\mathsf{S}}(n_1, u)$ as follows:

1. $\underline{A}^{H,\Sigma}(n_1)$ picks $u \leftarrow \{0,1\}^m$ and $p_0 \leftarrow \mathcal{P}$ and finds $y_{p_0} \leftarrow \texttt{genShare}^{F(u,\cdot)}(p_0, n_1)$.
2. $\underline{A}$ then simulates $A^{H,\Sigma,\mathsf{S}}(n_1, u)$ and records all Σ-calls and H-calls made by A.
3. If A calls $\mathsf{S}(m, y)$, then $\underline{A}$ checks if $P(m)^{y_{p_0}} \equiv y \pmod{n_1}$ and in case of match:

– If A has previously made a call $\sigma \leftarrow \Sigma(m)$, then $\mathsf{S}(m, y)$ is replied with σ. We say that such an S-call is *repeating*, otherwise the call is *non-repeating*.
– If A did not make the call $\sigma \leftarrow \Sigma(m)$ and made a call $m \leftarrow H(M)$, then $\underline{A}$ stops and outputs (M, σ), that is a successful existential forgery.
– If A did not make a call $H(m)$, then $\underline{A}$ makes a Σ-call $\sigma \leftarrow \Sigma(m)$ and answers $\mathsf{S}(m, y)$ with σ.

If there is no match, $\underline{A}$ increments the wrong-password counter and if the counter reaches to the limit T, no S-calls are answered any more.

The running time of $\underline{A}$ does not exceed tt_{ex} because the only overhead comes from the simulation of S-calls where one exponentiation is done in each call.

Hence, the probability that A succeeds without making any successful non-repeating S-calls does not exceed $\frac{tt_{\text{ex}}}{S}$.

The probability that A succeeds with an S-call (m, y) so that before this S-call, A did not make any H-call with output m (such as $m \leftarrow H(M)$), does not exceed $\frac{t^2}{2^h}$. This is because the number of S-calls with such m-s is limited by the running time t and for every H-call $m' \leftarrow H(M')$, the probability that m' belongs to the set of m-s that have been inputs of S-calls made before calling $m' \leftarrow H(M')$ is limited to $\frac{t}{2^h}$.

The probability that A ever makes an S-call (m, y) such that the call $m \leftarrow H(M)$ was made prior to the S-call (m, y) and the period $\operatorname{ord}(m)$ of element $P(m)$ is less than s is by Definition 6 limited to $t \cdot \pi^P_{p,q}(s)$.

The probability that A ever makes an S-call (m, y) such that the call $m \leftarrow H(M)$ was made prior to the S-call (m, y) with $\operatorname{ord}(m) \geq s$ and for which there are two different passwords $p, p' \in \mathcal{P}$ with $m^{y_p} \equiv m^{y_{p'}} \pmod{n}$ (where $y_p = \texttt{genShare}^{F(u,\cdot)}(p, n)$ and $y_{p'} = \texttt{genShare}^{F(u,\cdot)}(p', n)$) is by Lemma 4 limited to $\frac{K^2}{2s} + \frac{K}{S}(t_{\text{ex}} + \log_2 K)$.

The probability that A succeeds with an S-call while all the S-calls (m, y) are such that $\{P(m)^{y_p} \bmod n_1\}_{p\in\mathcal{P}}$ are all different equals to the probability of guessing the correct password, which does not exceed $\frac{T}{K}$. Hence, the success probability of A is

$$\delta \leq \frac{T}{K} + t \cdot \frac{K^2}{2s} + t \cdot \frac{K}{S'}(t_{\mathsf{ex}} + \log_2 K) + t \cdot \pi^P_{p,q}(s) + \frac{t^2}{2^h} + \frac{tt_{\text{ex}}}{S} \quad .$$

□

Lemma 3. *If* $a, b \in \mathbb{Z}_n$, $v \geq \varphi(n)$, *and* $y \leftarrow \mathbb{Z}_v$, *then* $\Pr[a^y \equiv b \pmod{n}] \leq \frac{1}{\operatorname{ord}(a)}$.

Proof. If $b \notin \langle a \rangle$, i.e. if b is not in the subgroup generated by a, then the probability is 0. If $b = a^c$, where $0 \leq c < \operatorname{ord}(a)$, then there are no more than $\frac{v}{\operatorname{ord}(a)}$ values of y, such that $a^y \equiv b \pmod{n}$. Indeed, $a^y \equiv b \pmod{n}$ is equivalent to $y = c + k \cdot \operatorname{ord}(a)$ and from $0 \leq d < v$, we get $0 \leq k < \frac{v-c}{\operatorname{ord}(a)} \leq \frac{v}{\operatorname{ord}(a)}$. Hence, $\Pr[a^y \equiv b \pmod{n}] \leq \frac{1}{v} \cdot \frac{v}{\operatorname{ord}(a)} = \frac{1}{\operatorname{ord}(a)}$. □

Lemma 4. *Let* $m \in \mathbb{Z}_n$ *and* $\operatorname{ord}(m) \geq s$. *Let* $F(u, \cdot)$ *be an* S*-secure PRF. For every password* $p \in \mathcal{P}$, *let* $y_p = \texttt{genShare}^{F(u,\cdot)}(p, n)$. *Then, the probability* δ *of*

Algorithm 5. Distinguisher D_n^{Φ} for $F(u,\cdot)$.

```
$Y \leftarrow \emptyset$;
for every $p \in \mathcal{P}$ do
    $y_p \leftarrow \texttt{genShare}^{\Phi}(p,n)$;
    if $y_p \in Y$ then
        return 1;
    else
        $Y \leftarrow Y \cup \{y_p\}$;
return 0;
```

having $p, p' \in \mathcal{P}$ such that $p \neq p'$ and $P(m)^{y_p} \equiv P(m)^{y_{p'}} \pmod{n_1}$ does not exceed $\frac{K^2}{2s} + \frac{K}{S}(t_{\mathrm{ex}} + \log_2 K)$.

Proof. By Lemma 3, if $\{y_p\}_{p\in\mathcal{P}}$ were pairwise independent, then for any fixed pair $p \neq p'$: $\mathrm{Prob}\left[m^{y_p} \equiv m^{y_{p'}} \pmod{n_1}\right] \leq \frac{1}{s}$. As there are no more than $K^2/2$ such pairs, the probability of having such a pair with $P(m)^{y_p} \equiv P(m)^{y_{p'}}$ does not exceed $\frac{K^2}{2s}$. Consider now the next distinguisher D_n^{Φ} for $F(u,\cdot)$ (Algorithm 5).

By definition, $\delta = \underset{u}{\mathrm{Prob}}\left[D^{F(u,\cdot)} = 1\right]$. If $\mathcal{F}$ is a random oracle, then $\{y_p\}_{p\in\mathcal{P}}$ are pairwise independent and hence $\underset{\mathcal{F}}{\mathrm{Prob}}\left[D^{\mathcal{F}} = 1\right] \leq \frac{K^2}{2s}$. The running time of D includes the computation time Kt_{ex} of $\{y_p\}_{p\in\mathcal{P}}$ and the search time $K\log_2 K$ for checking that $y_p \in Y$ and hence, $\delta \leq \frac{K^2}{2s} + \frac{K}{S}(t_{\mathrm{ex}} + \log_2 K)$ by the S-security of $F(u,\cdot)$. □

Theorem 4. *If p, q are (ℓ, s)-safe primes, $P\colon \{0,1\}^h \to \mathbb{Z}_{pq}$ is a padding function ($h < pq$), then*

$$\pi_{p,q}^{P}(s) \leq \frac{16\ell^4 + 4\ell + 1}{2^h} \leq \frac{16(\ell+1)^4}{2^h} = 2^{4\log_2(\ell+1)+4-h} \ .$$

Proof. As P is injective, there are 2^h possible values of $P(m)$ which due to the uniform distribution of m, these values are uniformly distributed in the image of P as a 2^h-element subset of $\mathbb{Z}_{pq}$. By Lemmas 5 and 6, the number of elements in $\mathbb{Z}_{pq}$ with order less than s does not exceed $16\ell^4 + 4\ell + 1$. □

Lemma 5. *If p, q are (ℓ, s)-safe primes, there are at most $16\ell^4$ elements $m \in \mathbb{Z}_{pq}^*$ with $\mathrm{ord}(m) < s$.*

Proof. By assumptions, there are prime numbers $p_1', \ldots, p_k', q_1', \ldots, q_k' \geq s$ so that $p - 1 = 2ap_1' \ldots p_k'$ and $q - 1 = 2a'q_1' \ldots q_k'$, where both a and a' belong to the interval $[1 \ldots \ell{-}1]$. Hence, the size of the group $\mathbb{Z}_{pq}^*$ is $\varphi(pq) = (p{-}1)(q{-}1) = 4aa'p_1' \ldots p_k'q_1' \ldots q_k'$. As the order of an element must be a divisor of the size of the group, any element m of $\mathbb{Z}_n^*$ has order $\mathrm{ord}(m)$ that divides $4aa'$ or is divisible by one of the primes p_i' or q_i' which means $\mathrm{ord}(m) \geq s$. As all the elements of orders dividing $4aa'$ are roots of the polynomial $X^{4aa'} - 1$ in $\mathbb{Z}_{pq} \cong \mathbb{Z}_p \times \mathbb{Z}_q$

and any polynomial of degree d may have no more than d roots in $\mathbb{Z}_p$ and $\mathbb{Z}_q$, the number of roots in $\mathbb{Z}_n$ cannot exceed d^2. Hence, the number of elements of degree less than s does not exceed $d^2 = (4aa')^2 \leq 16\ell^4$. □

Lemma 6. *If p, q are (ℓ, s)-safe primes, there are at most $4\ell + 1$ elements $m \in \mathbb{Z}_{pq} \backslash \mathbb{Z}_{pq}^*$ with* $\text{ord}(m) < s$.

Proof. As in the previous lemma, let $p - 1 = 2ap'_1 \ldots p'_k$ and $q - 1 = 2a'q'_1 \ldots q'_k$. An element of $\mathbb{Z}_{pq}$ is non-invertible (i.e. $\in \mathbb{Z}_{pq} \backslash \mathbb{Z}_{pq}^*$) if and only if it is divisible by p or q. As $\mathbb{Z}_{pq} \cong \mathbb{Z}_p \times \mathbb{Z}_q$, the non-invertible elements are represented by pairs $(0, m')$ and $(m', 0)$. An order of $(m', 0)$ in $\mathbb{Z}_{pq}$ is hence the same as the order of m' in the field $\mathbb{Z}_p$. As the order of an element $m' \neq 0$ must divide $p - 1 = 2ap'_1 \ldots p'_k$, then either the order divides one of p'_i and is therefore at least s, or $\text{ord}(m')$ divides $2a$ and hence m' is a root of the polynomial $X^{2a} - 1 = 0$ in $\mathbb{Z}_p$. Hence, there are at most $2a \leq 2\ell$ elements $m' \neq 0$ with $\text{ord}(m') < s$ in $\mathbb{Z}_p$. Hence, there are at most 2ℓ non-zero elements of $\mathbb{Z}_{pq}$ divisible by p that have order less than s. The same can be said about the elements divisible by q. Hence, together with 0 there are at most $4\ell + 1$ elements in $\mathbb{Z}_{pq} \backslash \mathbb{Z}_{pq}^*$ with order less than s. □

5.4 Security Against Memory Read

If the adversary A has accessed the memory of the device either during signing or key generation, then it may have obtained the client's share d'_1 of client's private exponent d_1 (either directly or by computing it from (u, pwd)). Possibly it has also learned server's share d''_1. Additionally, A has learned the one-time password r. The knowledge of (d'_1, r) is sufficient for A to masquerade the legitimate client. This is possible until the next query by the client. There are two possibilities.

1. A has changed the one-time password in the meantime. As the client presents an old one-time password, the server deactivates the client.
2. The one-time password is still valid. In this case, the client is served, and the one-time password is changed to a uniformly randomly distributed value which A does not know and can guess it with success probability of only $2^{-\eta}$. Hence with probability $(1 - 2^{-\eta})$, the next adversarial query will be ignored and the client will be deactivated.

6 An Instantiation of Security Parameters

Let us have a system with the following parameters:

- We use RSA-2048 with $(2^{16}, 2^{200})$-safe primes ($\ell = 2^{16}$, $s = 2^{200}$) and assume it to have 112 bits of security, i.e. $S = 2^{112} t_{\text{bl}}$
- We use AES-128 as the building block F in the PRF ($m = 128$, $q = \frac{2048}{128} = 16$) and assume AES-128 to have 128 security bits as a PRF. Then $S' \approx 2^{124}$.
- The time for a public exponentiation is $t_{\text{pe}} \approx 2^9 \cdot t_{\text{bl}}$.

- The time for a private exponentiation is $t_{\mathrm{ex}} \approx 2^{13} \cdot t_{\mathrm{bl}}$.
- There are $K = 2^{30}$ passwords, and we accept $T_0 = 8 = 2^3$ wrong trials.
- We use a 256-bit hash function ($h = 256$).

Thus, $\frac{T_0}{K} \approx 2^{-27}$ and $\pi^P_{p,q}(s) \approx 2^{-188}$, $\frac{K^2}{2s} \approx 2^{-141}$, $\frac{K}{S'}(t_{\mathsf{ex}} + \log_2 K) \approx 2^{-81}$, $\frac{t_{\mathrm{ex}}}{S} \approx 2^{-99}$, i.e. we have 99 bits of security and 98 bits against malicious servers.

If a *Device Read* occurs, the adversary has to spend 2^{54} time units for doubling its guessing chances (compared to $\frac{T_0}{K} \approx 2^{-27}$). By using AES-256, the necessary time for an adversary to double the guessing chances will be 2^{183}.

7 Practical Implementation

Generating safe primes p (where $\frac{p-1}{2}$ is also prime) is time-consuming, especially in low-power mobile devices. Hence we have settled with (ℓ, s)-safe primes, with slight loss in security reductions (Sect. 6), but with much faster generation. For example, for a 1024-bit $p = 2ap' + 1$, by using 15-bit a (with $2^{14} \leq a < 2^{15}$), we need a 1008-bit p'. Complete signing uses three RSA operations, one in the client's device and two in the server. Additionally, the server needs to perform at least one RSA verification. RSA signing operations in the app take tens of milliseconds (for example, on the Nexus 5X, about 30 ms). However, more significant is the network delay to transmit the signature share from the app to the server, which may take up to 100 or 1000 ms. All together, the performance of the complete signing operation for the authentication and digital signatures, is still reasonable and sufficiently user friendly. RSA key generation takes seconds for 2048-bit client's modulus (n_1) and tens of seconds for 3072-bit modulus. For example, on the Nexus 5X, it takes about 3 s to generate the key for 2048-bit modulus and about 17 s for 3072-bit modulus. The practicality of Smart-ID scheme has been demonstrated by its deployment. It became publicly available in Estonia, Latvia, and Lithuania in early November 2016[1]. By June 2017, Smart-ID had more than 200,000 registered users across the three states[2].

References

1. Bellare, M., Rogaway, P.: The exact security of digital signatures-how to sign with RSA and rabin. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996). doi:10.1007/3-540-68339-9_34
2. Bellare, M., Sandhu, R.: The security of practical two-party RSA signature schemes. Cryptology e-print archive 2001/060
3. Blanchet, B.: Modeling and verifying security protocols with the applied Pi calculus and ProVerif. Found. Trends Priv. Secur. **1**(1–2), 1–135 (2016)
4. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 425–439. Springer, Heidelberg (1997). doi:10.1007/BFb0052253

[1] https://sk.ee/en/News/sk-introduced-the-new-e-identity-solution-smart-id.

[2] https://sk.ee/en/News/number-of-smart-id-users-in-the-baltics-surpasses-200-000.

5. Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys. J. ACM **48**(4), 702–722 (2001)
6. Boneh, D., Horwitz, J.: Generating a product of three primes with an unknown factorization. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 237–251. Springer, Heidelberg (1998). doi:10.1007/BFb0054866
7. Camenisch, J., Lehmann, A., Neven, G., Samelin, K.: Virtual smart cards: How to sign with a password and a server. In: Zikas, V., Prisco, R. (eds.) SCN 2016. LNCS, vol. 9841, pp. 353–371. Springer, Cham (2016). doi:10.1007/978-3-319-44618-9_19
8. Damgård, I., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 152–165. Springer, Heidelberg (2001). doi:10.1007/3-540-44987-6_10
9. Damgård, I., Mikkelsen, G.L.: On the theory and practice of personal digital signatures. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 277–296. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00468-1_16
10. Damgård, I., Mikkelsen, G.L.: Efficient, robust and constant-round distributed RSA key generation. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 183–200. Springer, Heidelberg (2010). doi:10.1007/978-3-642-11799-2_12
11. Damgård, I., Mikkelsen, G.L., Skeltved, T.: On the security of distributed multi-prime RSA. In: Lee, J., Kim, J. (eds.) ICISC 2014. LNCS, vol. 8949, pp. 18–33. Springer, Cham (2015). doi:10.1007/978-3-319-15943-0_2
12. Desmedt, Y.: Society and group oriented cryptography: a new concept. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (1988). doi:10.1007/3-540-48184-2_8
13. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, New York (1990). doi:10.1007/0-387-34805-0_28
14. Frankel, Y., MacKenzie, P.D., Yung, M.: Robust efficient distributed RSA key generation. In: Vitter, J.S. (ed.) STOC, pp. 663–672. ACM (1998)
15. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust and efficient sharing of RSA functions. J. Cryptol. **13**, 273–300 (2000)
16. Gilboa, N.: Two party RSA key generation. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 116–129. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_8
17. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold paillier in the two-party setting. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 313–331. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27954-6_20
18. Kwon, T.: On the difficulty of protecting private keys in software. In: Chan, A.H., Gligor, V. (eds.) ISC 2002. LNCS, vol. 2433, pp. 17–31. Springer, Heidelberg (2002). doi:10.1007/3-540-45811-5_2
19. Kwon, T.: Robust software tokens – Yet another method for securing user's digital identity. In: Safavi-Naini, R., Seberry, J. (eds.) ACISP 2003. LNCS, vol. 2727, pp. 476–487. Springer, Heidelberg (2003). doi:10.1007/3-540-45067-X_41
20. Luby, M.: Pseudorandomness and Cryptographic Applications. Princeton University Press, Princeton (1996)
21. MacKenzie, P., Reiter, M.K.: Networked cryptographic devices resilient to capture. Int. J. Inf. Secur. **2**(1), 1–20 (2003)
22. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)

23. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000). doi:10.1007/3-540-45539-6_15
24. Smart, N.P. (ed.): Algorithms, Key Size and Protocols Report. Deliverable D5.2 of ECRYPT CSA, 17 October 2016
25. RSA Laboratories. PKCS #1: RSA Encryption Standard, ver. 2.2, October 2012