Cryptanalysis and Improvement of the Smart–ID Signature Scheme

Augustin P. SARR

LACCA, UFR SAT, Université Gaston Berger de Saint–Louis,
Saint–Louis, Senegal

Abstract. At ESORICS 2017, Buldas *et al.* proposed an efficient (software only) server supported signature scheme, geared to mobile devices, termed Smart–ID. A major component of their design is a clone detection mechanism, which allows a server to detect the existence of clones of a client's private key share. We point out a flaw in this mechanism. We show that, under a realistic race condition, an attacker which holds a password camouflaged private share can lunch an online dictionary attack such that (*i*) if all its password guesses are wrong, it is very likely that the attack will not be detected, and (*ii*) if one of its guesses is correct, it can generate signatures on messages of its choice, and the attack will *not* be detected. We propose an improvement of Smart–ID to thwart the attack we present.

Keywords: Smart–ID, four–prime RSA, mobile devices, clone detection, undetectable online dictionary attack.

1 Introduction

Digital signatures are used in every day communications and commerce. Given the widespread use of mobile devices, and the issue of a secure storage of the private keys, server supported software only solutions seem to be an interesting approach. Companies are now deploying software only threshold cryptography for key protection on mobile devices. Cyberetica [4] proposes an authentication and digital signature platform based on Smart–ID [1]. Smart–ID is a signature scheme, a modification of Damgård *et al.*'s four prime RSA [2], geared to software only implementations on mobile devices.

A Smart–ID private key is shared between the device and a server, in a way to avoid the existence of a reference point for offline dictionary attacks, at both the device and the server. In addition, the signature generation integrates a clone detection mechanism which is claimed to allow the server to detect the existence of clones of a client's private share. The number of Smart–ID users grew from 200,000 in 2017 [1] to 1,800,000 in 2019 [3].

Unfortunately, as we show in Section 2, there is a subtle flaw in the clone detection mechanism; this invalidates some of the claimed security attributes. Namely, we show that, under a plausible race condition, an attacker which holds a clone of the client's password camouflaged private key share (recall that the implementation is software–only) can issue up to $(T_0 - 1)$ *undetectable* password guesses, where T_0 is the maximum wrong password guesses the service allows. Moreover, if one of the guesses is correct the attacker may generate signatures on messages of its choice, and this will *not* be detected. We improve the clone detection mechanism to thwart the attack we present.

This paper is organized as follows. In Section 2 we recall the Smart–ID scheme, then we present an attack which invalidates the security of the clone detection mechanism. In Section 3, we propose an improved variant of Smart–ID, which thwarts the attack we present. We provide some concluding remarks in Section 4.

We use the following notations. If S is a set, $a \leftarrow_{\mathrm{R}} S$ means that a is chosen uniformly at random from S. A prime number p is said to be (l, s)–safe if $p = 2ap'_1 \cdots p'_k$ where $p'_{i, i \in \{1, \cdots k\}}$ are primes greater than s and $1 \leqslant a \leqslant l$. For an integer n, $[n]$ denotes the set $\{0, \cdots, n\}$. If n_1 and n_2 are such that $\gcd(n_1, n_2) = 1$, $\mathsf{crt}((\sigma_1, n_1), (\sigma_2, n_2))$ refers to the unique $\sigma \in [n_1 n_1 - 1]$ such that $\sigma = \sigma_1 \mod n_1$ and $\sigma = \sigma_2 \mod n_2$.

2 Attacking the Smart–ID Signature Scheme

2.1 Description of the Scheme

Given a security parameter η, the Setup algorithm defines an RSA modulus length k, suitable values for l and s, a public exponent e, and a pseudorandom function $\mathsf{GenShare}$ which takes as inputs $u \in \{0,1\}^\eta$, a password $\mathsf{pwd} \in \{0,1\}^l$ and an RSA modulus n_1 and outputs $d_1 < n_1$. It defines also a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\eta_\mathsf{H}}$, with $\eta_\mathsf{H} \leqslant (k-1)$, a padding scheme P, and an upper bound T_0 on the number of password attempts a user may perform.

Shared key generation. Assuming a secure channel between the client clt and the server srv, clt generates two (l, s)–safe primes p_1 and q_1 (such that $\gcd(e, (p_1 - 1)(q_1 - 1)) = 1$) and computes $n_1 = p_1 q_1$, and $d_1 = e^{-1} \mod \phi(n_1)$. It generates $u, r \leftarrow_{\mathrm{R}} \{0,1\}^\eta$ and $d'_1 = \mathsf{GenShare}(u, \mathsf{pwd}, n_1)$, where pwd is the user's password, and $d''_1 = d_1 - d'_1 \mod \phi(n_1)$. Then, clt sends $\langle d''_1, n_1, r \rangle$ to srv.

At receipt of clt's message, srv generates two (l, s)–safe primes p_2 and q_2 (such that $\gcd(e, (p_1 - 1)(q_1 - 1)) = 1$) and computes $n_2 = p_2 q_2$, $d_2 = e^{-1} \mod \phi(n_2)$, and $n = n_1 n_2$. Then, it sends back n to clt and stores $\langle n_1, n_2, d''_1, d_2, r, T = T_0 \rangle$. At receipt of srv's message, clt stores $\langle n, n_1, u, r \rangle$ and safely deletes all the other values. The public key is $pk = (e, n)$.

Signature Generation. For a signature on $m' \in \{0,1\}^*$, clt computes $m = \mathsf{P}(\mathsf{H}(m'))$; then, from the user's password pwd, it derives $d'_1 = \mathsf{GenShare}(u, \mathsf{pwd}, n_1)$ and $y = m^{d'_1} \mod n_1$. It chooses $r' \leftarrow_{\mathrm{R}} \{0,1\}^\eta$ and sends $\langle y, m, r, r' \rangle$ to srv.

At receipt of $\langle y, m, r, r' \rangle$, srv verifies that clt is active; if so, it lookups the record $\langle n_1, n_2, d''_1, d_2, \hat{r}, T \rangle$. If $\hat{r} \neq r$, srv deactivates clt; else, it computes $\sigma_1 = y m^{d''_1} \mod n_1$ and $\hat{m} = \sigma_1^e \mod n_1$. If $\hat{m} \neq m$, it drops the request, decrements T, and deactivates clt in the case $T = 0$. If $\hat{m} = m$, it computes $\sigma_2 = m^{d_2} \mod n_2$, $\sigma = \mathsf{crt}((\sigma_1, n_1), (\sigma_2, n_2))$, sends back $\langle \sigma, m \rangle$ to clt, sets $T = T_0$, and stores $\langle n_1, n_2, d_1'', d_2, r', T \rangle$. At receipt of $\langle \sigma, m \rangle$, clt stores $\langle n, n_1, u, r' \rangle$. The signature on m' is σ.

Signature Verification. To verify a signature σ on m', with regard to a public key $pk = (n, e)$, one computes $m = \mathsf{P}(\mathsf{H}(m'))$ and verifies that $\sigma^e = m \mod n$.

The Clone Detection Mechanism. A major component in Smart–ID's design is its clone detection mechanism. During the key pair generation the client clt sends to the server not only a share d_1'' of d_1, the part of the private key it generates, but also a nonce r the server should expect to receive in clt's next service query. And, each time clt uses the services, it sends a new nonce r' together with r; from there srv expects to receive r' in clt's next query. An adversary $\mathcal{A}$ which holds a clone of clt's private share d_1 has to send a new nonce r' at each service query. Then, it is expected that the value $\mathcal{A}$ sends be different from the value r at the legitimate client clt. And then, the existence of the clone be detected when clt attempts to query the service.

Unfortunately, this analysis mistakenly assumes that an attacker which holds a clone of clt's private share (which may password camouflaged or not) will follow the protocol's description. In particular, the analysis assumes that $\mathcal{A}$ will choose r' uniformly at random from $\{0,1\}^\eta$. As we show, this seemingly insignificant shortcoming induces major weaknesses in the clone detection mechanism.

2.2 Undetectable Online Password Guesses

Assuming a realistic race condition, we show how an attacker $\mathcal{A}$ which holds a clone of clt's password camouflaged private share, and aims to have valid signatures on $m_1', m_2', \cdots, m_k' \in \{0,1\}^*$, can issue up to $(T_0 - 1)$ online password guesses such that (*i*) if all the guesses are wrong, it is very likely that the attack remains undetected, and (*ii*) if one of the guesses is correct, $\mathcal{A}$ generates signatures on the messages and the attack will *not* be detected. Clearly, this indicates a failure of Smart–ID's clone detection mechanism.

We assume that $\mathcal{A}$ obtains a clone of a client's password camouflaged share after a successful use of the service, so that $T = T_0$ at srv; $\mathcal{A}$ performs as in Algorithm 1.

Under the realistic assumption that the legitimate client clt does not use the service before the attack is completely executed, it is very likely that the attack remains undetected. In effect, $\mathcal{A}$ performs at most $(T_0 - 1)$ password guesses. And, if none of the guesses is correct, the device is not deactivated ($T = 1$, there remains one possible attempt). Now, as the attacker always used $r' = r$ as a next incoming nonce, the server still expects to receive a nonce with value r in the next request. This corresponds the nonce value at clt. So, it is very likely that when the legitimate device owner connects to the service, it derives the right private share d_1' and sends the nonce r, so that the value of T is set again to T_0,

Algorithm 1 Undetectable online password guesses

1) Computes $m_1 = \mathsf{P}(\mathsf{H}(m'_1))$, and recover the tuple $\langle n, n_1, u, r\rangle$ (which is stored unencrypted in the clone);
2) For each password $\mathsf{pwd}_i, i \in \{1, \cdots, T_0 - 1\}$, to test, do the following:
   a) Compute $\hat{d}'_{1,i} = \mathsf{GenShare}(u, \mathsf{pwd}_i, n_1)$ and $y_{1,i} = m_1^{\hat{d}'_{1,i}} \mod n_1$;
   b) Send $\langle y_{1,i}, m_1, r, r\rangle$ to srv;
   c) If srv responds with a pair $\langle \sigma_1, m_1\rangle$ such that $\sigma_1^e = m_1 \mod n$ then
      i) Store $\mathsf{pwd} = \mathsf{pwd}_i$ and $d'_1 = \mathsf{GenShare}(u, \mathsf{pwd}_i, n_1)$ as the right password and private key share, respectively;
      ii) For each $m_j = P(\mathsf{H}(m'_j))$, $j \in \{2, \cdots, k\}$:
         *ii*1) Compute $y_j = m_j^{d'_1} \mod n_1$ and send $\langle y_j, m_j, r, r\rangle$ to srv;
         *ii*2) At receipt of $\langle \sigma_j, m_j\rangle$, store σ_j as a signature on m'_j.

and the attacker's wrong password guesses remain undetected. In contrast, if the device owner mistakenly types a wrong password, the device is deactivated and the attack is detected. This event can be made rarer by reducing the number of password guesses the attacker performs (to $(T_0 - 2)$, for instance).

If one of the password guesses is correct, $\mathcal{A}$ which is now aware of d'_1 generates the signatures σ_j on m'_j for $j \in \{1, \cdots, k\}$. After the signature generations, the value of T at srv is T_0, and the value of r srv expects to receive is the one at clt. Hence, under the race condition that clt does not use the service before the attack is completely executed, the attack *will not be detected*.

3 Improving the Smart–ID Scheme

We propose a variant of Smart–ID, which resists to our attack.

At first glance, it may be tempting to modify the server to require that two consecutive nonces be different, *i. e.*, in a signature generation, the current nonces r and the next nonce r' a client provides be different. This modification is not enough, if $(T_0 - 1) \geqslant 3$ or if one of the password guesses is correct, as the attacker can use consecutive nonces $r = r_1, r_2, \cdots, r_L$, with $L \geqslant (T_0 - 1)$ ($L = T_0 - 1$ if all the guesses are wrong) such that $r_1 \neq r_2$, $r_2 \neq r_3, \cdots$, but $r_L = r$.

More generally, the server can require that consecutive T_0 nonces be pairwise different; this requirement induces no modification at the client (the probability of collision $\leqslant T_0^2/2^\eta$, which is negligible). Unfortunately, this change remains unsatisfactory, as if $\mathcal{A}$ succeeds in one of its guesses, it can query the service $L > T_0$ times, with nonces $r = r_1, r_2, \cdots, r_k$ such that $r_1 \neq r_2$, $r_2 \neq r_3, \cdots$, but $r_L = r$. The attack will not be detected.

A better approach is to modify the server so that it contributes to the nonce generation. In this way, it becomes infeasible for a malicious client to masquerade so that the nonce srv expects to receive holds a specific value. We describe hereunder the modified Smart–ID variant we obtain with such a modification. The *setup* and *signature verification* algorithms are the same as in original scheme.

Shared key generation. We assume a secure channel between the client clt and the server srv. The client generates two (l,s)–safe primes p_1 and q_1 and computes $n_1 = p_1 q_1$ and $d_1 = e^{-1} \mod \phi(n_1)$. It generates $u, r_c \leftarrow_{\mathrm{R}} \{0,1\}^\eta$ and computes $d'_1 = \mathsf{GenShare}(u, \mathsf{pwd}, n_1)$, where pwd is the user's password, and $d''_1 = d_1 - d'_1 \mod \phi(n_1)$. Then it sends $\langle d''_1, n_1, r_c\rangle$ to srv.

At receipt of clt's message, srv generates two (l,s)–safe primes p_2 and q_2; it computes $n_2 = p_2 q_2$, $d_2 = e^{-1} \mod \phi(n_2)$, $r_s \leftarrow_{\mathrm{R}} \{0,1\}^\eta$, and $n = n_1 n_2$. Then, it sends back $\langle n, r_s\rangle$ to clt and stores $\langle n_1, n_2, d''_1, d_2, r_c, r_s, T = T_0\rangle$. At receipt of srv's message, clt stores $\langle n, n_1, u, r_c, r_s\rangle$ and safely deletes all the other values. The public key is $pk = (e, n)$.

Signature Generation. For a signature on m', clt generates $m = \mathsf{P}(\mathsf{H}(m'))$. Then, it gets the user's password pwd, and derives $d'_1 = \mathsf{GenShare}(u, \mathsf{pwd}, n_1)$ and $y = m^{d'_1} \mod n_1$. It chooses $r'_c \leftarrow_{\mathrm{R}} \{0,1\}^\eta$ and sends $\langle y, m, r_c, r_s, r'_c\rangle$ to srv.

At receipt of $\langle y, m, r_c, r_s, r'_c\rangle$, srv verifies that clt is active. If so, it lookups the record $\langle n_1, n_2, d''_1, d_2, \hat{r}_c, \hat{r}_s, T\rangle$. It computes $\sigma_1 = y m^{d''_1} \mod n_1$ and $\hat{m} = \sigma_1^e \mod n_1$, and performs as follows.

- If $(\hat{r}_c, \hat{r}_s) \neq (r_c, r_s)$ and $\hat{m} \neq m$ then srv alerts[1] on the existence of a clone of the password camouflaged share of clt, and drops the request.
- If $(\hat{r}_c, \hat{r}_s) \neq (r_c, r_s)$ and $\hat{m} = m$ then srv deactivates the client (there is probably a clone of clt's private share).
- If $(\hat{r}_c, \hat{r}_s) = (r_c, r_s)$ and $\hat{m} \neq m$ then
  - srv chooses $r'_s \leftarrow_{\mathrm{R}} \{0,1\}^\eta$, and sends ("0", r'_s) to clt;
  - it decrements T, stores $\langle n_1, n_2, d''_1, d_2, r'_c, r'_s, T\rangle$, and deactivates clt in the case $T = 0$.
- If $(\hat{r}_c, \hat{r}_s) = (r_c, r_s)$ and $\hat{m} = m$ then srv computes $\sigma_2 = m^{d_2} \mod n_2$, $\sigma = \mathsf{crt}((\sigma_1, n_1), (\sigma_2, n_2))$, chooses $r'_s \leftarrow_{\mathrm{R}} \{0,1\}^\eta$, and sends back $\langle \sigma, m, r'_s\rangle$ to clt. It sets $T = T_0$ and stores $\langle n_1, n_2, d''_1, d_2, r'_c, r'_s, T\rangle$.

At receipt of srv's message, clt performs as follows.

- If the message parses as ("0", r'_s) then clt stores $\langle n, n_1, u, r'_c, r'_s\rangle$ (the user probably typed a wrong password and the signature generation failed).
- Else (the message parses as $\langle \sigma, m, r'_s\rangle$),
  - clt stores $\langle n, n_1, u, r'_c, r'_s\rangle$;
  - the signature on m' is σ.

Clone Detection. By defining the nonce a client provides, when using the service, as a pair (r_c, r_s) such that r_c is generated by the client and r_s by the server, neither the server nor the client can masquerade so that a nonce takes a specific value. In this way, once an attacker (which holds either a clone of the password camouflaged share or a clone of clt's private share) uses the service, through an online password guess or a signature generation, the nonce the server expects for the next query changes. And, except with negligible probability, it becomes different from the one at clt. Thereby, the existence of the clone will be detected the next time clt uses the service.

[1] There may exist a secondary channel between srv and clt.

Remark 1. a) The clone detection mechanism may be of interest in other client–server settings.

b) To reduce the communication cost of a signature generation, the nonce clt sends can be defined to be $\mathsf{H}'(r_c, r_s)$, for some cryptographic hash function $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^\eta$, instead of (r_c, r_s).

4 Concluding Remarks

The Smart–ID scheme, built from Damgård *et al.*'s four prime RSA, is geared to server supported software only implementations on mobile devices. While this approach provides an interesting solution for the issue of a secure storage of the private keys on mobile devices, it yields easily clonable (software only) applications. To mitigate this issue, the Smart–ID design integrates a clone detection mechanism.

We pointed out a subtle shortcoming in the clone detection mechanism and showed, under a realistic race condition, how an attacker which holds a clone of the password camouflaged private share of a client can lunch online password guesses such that (*i*) if all the guesses are wrong, it is likely that the attack will no be detected, and (*ii*) if one of the guesses is correct, the attacker may generate signatures on messages of its choice, while the attack will not be detected. We proposed a variant of Smart–ID which resists the attack we present.

It would be a nice feature of the Smart–ID scheme if device owners had the possibility to update their passwords. In a forthcoming stage, we will explore the question of password updates in password–based server supported signature schemes.

References

1. Buldas A., Kalu A., Laud P., Oruaas M.: Server-Supported RSA Signatures for Mobile Devices. In: Foley S., Gollmann D., Snekkenes E. (eds) Computer Security — ESORICS 2017. ESORICS 2017. LNCS, vol 10492. Springer, Cham (2017)
2. Damgård, I., Mikkelsen, G. L., Skeltved, T.: On the security of distributed multi-prime RSA. In: Lee J., Kim J. (eds) Information Security and Cryptology — ICISC 2014. ICISC 2014. LNCS, vol 8949. Springer, Cham (2015)
3. van de Poll M., Kalu A.: SplitKey — A Threshold Cryptography Case Study. NIST Threshold Cryptography Workshop 2019. Gaithersburg, MD, March 2019.
4. Cybernetica https://cyber.ee.