Cautionary note for protocol designers: Security proof is not enough

Aurore Gillet[1], Marc Joye[2] and Jean-Jacques Quisquater[1]

[1]UCL Crypto Group, Microelectronics Labs, University of Louvain
Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium
Email: jjq@dice.ucl.ac.be

[2]UCL Crypto Group, Dept of Mathematics, University of Louvain
Chemin du Cyclotron 2, B-1348 Louvain-la-Neuve, Belgium
Email: joye@agel.ucl.ac.be

URL: http://www.dice.ucl.ac.be/crypto/

1 Introduction

Security requirements for real-life applications still increase. For this purpose, many people develop cryptographic protocols in order to fulfill these needs.

When someone designs a new protocol, he usually uses cryptographic tools for which he is confident. The design and the analysis of these tools is the domain of the mathematician and the cryptographer. In many cases, a cryptoalgorithm is proved as secure as solving hard problems (e.g., factoring large composite numbers). No such "proofs" exist for the cryptanalysis of protocols. By definition, a protocol designer must suspect everything. The use of strong cryptoalgorithms is not sufficient to guarantee the security of a protocol. In some situations, a protocol may be completely subverted without compromising the security of the underpinning cryptoalgorithm. Such situations are called *protocol failures* [11].

Protocols are for example designed in order to establish a session key, to authenticate a transaction, to sign a document, etc... This is usually achieved by exchanging some messages between two or several people. In the sequel, we will show a failure on the most widely used public-key cryptosystem, the so-called RSA [13]. It may be briefly described as follows. Suppose that Alice wants to send a message m to Bob. To setup the system, Bob carefully selects two large primes p and q, computes $n_B = pq$, chooses a encryption key e_B relatively prime to $\phi(n_B)$, and computes the decryption key

d_B according to $e_B d_B \equiv 1 \pmod{\phi(n_B)}$. The public key of Bob is the pair (n_B, e_B) and his secret key is d_B. To send m to Bob, Alice forms the ciphertext $c = m^{e_B} \bmod n_B$ and sends it to Bob. Then, Bob recovers the plaintext by computing $m = c^{d_B} \bmod n_B$ with his secret key d_B. The security of this scheme relies on the intractability to factor n_B. The aim of a pirate, say Carol, is to recover m. This can be done as follows. Carol chooses a random number k, intercepts c, replaces it by $c' = ck^{e_B} \bmod n_B$, and sends c' to Bob. Now, when Bob deciphers c', he obtains $m' \equiv (c')^{d_B} \equiv mk \pmod{n_B}$. Since m' is meaningless, Bob discards it. If Carol can get access to this discard, she finds $m = m'k^{-1} \bmod n_B$. This failure was first pointed out by Davida [5]. Avoiding this attack is easy, the users have to really destroy the discards, or in other words to protect their bins. The lesson of this failure is that the protocol designer has to pay attention to what is generally accepted but not explicitly stated.

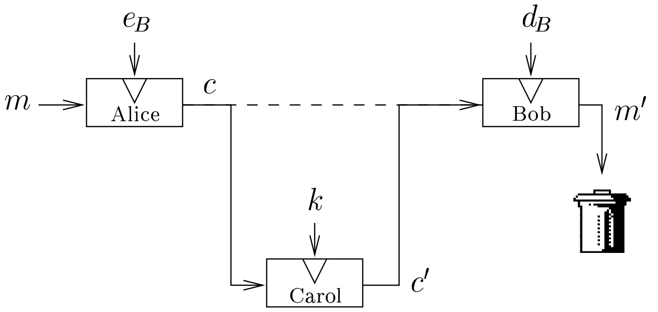

Figure 1: Davida's attack.

The decryption process can be speeded up by the Chinese Remainder Theorem (CRT). From p and q, Bob computes $m_p = c^{d_B \bmod p-1} \bmod p$ and $m_q = c^{d_B \bmod q-1} \bmod q$, and finally finds $m = \mathrm{CRT}(m_p, m_q)$ [12]. Suppose that Carol induces an external constraint on the deciphering device of Bob (e.g., ionizing or microwave radiation). Assume that the computation of m_p is correctly performed but not computation of m_q. So, Bob gets $m' = \mathrm{CRT}(m_p, m'_q)$ instead of m. If Bob discards m' and if Carol can get access to m', then she finds the secret factor p by computing $\gcd((m')^{e_B} - c \bmod n_B, n_B)$. Hence, $q = n_B/p$ and Carol can compute the secret decryption key d_B [10, 7].

This second attack is more dangerous because it completely breaks the system. This shows clearly the importance of checking cryptographic protocols for faults [3]. Note also that if Bob protects his bin, the attack does

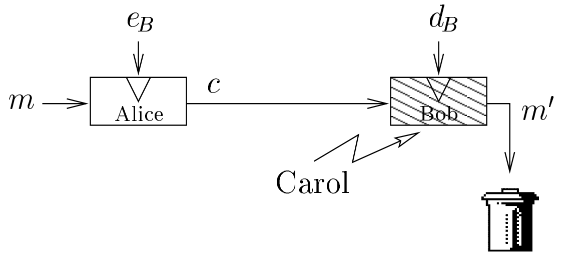

Figure 2: Lenstra's attack.

not remain applicable.

The two previous attacks show that it is extremely difficult for a protocol designer to determine whether his protocol is sound, even for very simple protocols. Some researchers proposed formal techniques for analyzing the soundness of protocols, such as the BAN logic [4] or the three systems presented in [8].

Another approach for the protocol designer is to try to find flaws in his protocol with all his experience of good and bad practice. In [1], Abadi and Needham give general rules helping protocol designers to avoid many of the pitfalls (see also [2]). In this paper, we push their work further, and show that the protocol designers must also take care about the hardware and software implementation of the protocol.

2 Cycling attack against faulty hardware

We shall illustrate this attack in the RSA signature context. Imagine that Alice wants to send a signed message to Bob. For this purpose, she carefully chooses two large primes p and q, and publishes their product $n = pq$. She also chooses a public verification key v according to $\gcd(v, \phi(n)) = 1$. The secret signature key s is computed so that $sv \equiv 1 \pmod{\phi(n)}$. Then, to sign a message m, Alice computes $S = m^s \bmod n$, and sends the pair (m, S) to Bob. To verify that S effectively is the signature of Alice corresponding to m, Bob checks whether $S^v \equiv m \pmod{n}$, where v is the public verification key of Alice.

We shall see that if the hardware is damaged, then a pirate can obtain some bits of the secret key s. Note that we do not deal with Chinese remaindering based implementations, because, as shown before, in this case one faulty computation modulo p or modulo q gives the secret factors of n.

2.1 Fast exponentiation algorithms

Suppose that you have to compute $S = m^s \bmod n$. If $s = \sum_{i=0}^{t-1} s_i 2^i$ is the binary decomposition of s, then this computation can efficiently be performed as depicted on Fig. 3. Indeed, if we respectively let $z^{(i)}$ and $y^{(i)}$ the values of z and y at step i (Lines 5 and 7 of Fig. 3), we have $z^{(i)} = z^{(i-1)} \left(y^{(i-1)}\right)^{s_{i-1}} \bmod n$ and $y^{(i)} = m^{2^i} \bmod n$. So,

$$\begin{aligned} S &\equiv m^{\sum_{i=0}^{t-1} s_i 2^i} \equiv \prod_{i=0}^{t-1} \left(m^{2^i}\right)^{s_i} \equiv \prod_{i=1}^{t} \left(y^{(i-1)}\right)^{s_{i-1}} \\ &\equiv \prod_{i=1}^{t} \frac{z^{(i)}}{z^{(i-1)}} \equiv z^{(t)} \pmod{n}. \end{aligned} \tag{1}$$

```
1. z := 1
2. y := m
3. for i := 1 to t do
4.     if (s_{i-1} = 1) then
5.         z := z * y mod n
6.     fi
7.     y := y * y mod n
8. od
9. S := z
```

Figure 3: Square-and-multiply method (I).

```
z := 1
for i := t - 1 downto 0 do
    if (s_i = 1) then
        z := m * z * z mod n
    else
        z := z * z mod n
    fi
od
S := z
```

Figure 4: Square-and-multiply method (II).

Scanning the bits of exponent s from left to right, we obtain another efficient algorithm to compute $S = m^s \bmod n$ (see Fig. 4). The drawback in this algorithm is that it cannot easily be parallelized. Consequently, we only focus on the first algorithm.

2.2 Our attack

Let $s = \sum_{i=0}^{t-1} s_i 2^i$ be the secret key to be discovered. We suppose that exponentiations are achieved by the right-to-left square-and-multiply algorithm (Fig. 3). The attack supposes that the register containing the value of y has some permanently damaged known bits, always '0' or '1'. Let $\mathcal{C}$ and $\mathcal{I}$ denote the subsets of correct and incorrect (i.e. damaged) bits of register y, respectively. So, any value $\hat{y}$ stored in register y can be written as*

$$\hat{y} = \sum_{j \in \mathcal{C}} \hat{y}_j 2^j + \sum_{j \in \mathcal{I}} \hat{y}_j 2^j. \tag{2}$$

From Eq. (1), the faulty signature corresponding to message m is given by

$$\hat{S} = \widehat{\mathbb{S}}(m) = \widehat{\mathbb{S}}(\hat{y}^{(0)}) = \prod_{i=0}^{t-1} \left(\hat{y}^{(i)}\right)^{s_i} \bmod n. \tag{3}$$

Let $f : \mathcal{C} \to \mathcal{C}$ be the function that maps the correct bits of $\hat{y}^{(i)}$ to the correct bits of $\hat{y}^{(i+1)}$. Since $\mathcal{C}$ is finite, the sequence must eventually cycle. The length μ of the tail and the length λ of the cycle can efficiently be computed by the Floyd's algorithm [9, exercise 6 on p. 7]. This algorithm is also known as the *kangaroos' method*. It has the advantage of minimizing the storage requirements.

Two kangaroos K_1 and K_2 cover the sequence generated by f. Kangaroo K_1 progresses in bounds of 1 unit and kangaroo K_2 in bounds of 2 units. Since the sequence cycles, the two kangaroos will meet. If they meet after k bounds, then $\hat{y}^{(k)} = \hat{y}^{(2k)}$. Once k has been found, it suffices to generate $\hat{y}^{(j)}$ and $\hat{y}^{(k+j)}$ for $j \geq 0$. Then, μ is the smallest integer j such that $\hat{y}^{(k+j)} = \hat{y}^{(j)}$.

Proof. Since $\hat{y}^{(k)} = \hat{y}^{(2k)}$, it follows that $(2k-k)$ is a multiple of λ. Moreover since $\hat{y}^{(\lambda+\mu)} = \hat{y}^{(\mu)}$, μ is the smallest integer j such that $\hat{y}^{(k+j)} = \hat{y}^{(j)}$. □

The length λ of the cycle is the smallest integer j such that $\hat{y}^{(k+j)} = \hat{y}^{(k)}$. Putting all together, we obtain the Floyd's algorithm (see Fig. 5).

To recover the bits of the secret exponent s, the pirate Carol proceeds as follows. She first chooses a random message m. Then, by the kangaroos' method, she computes the tail's length μ and the cycle's length λ of the sequence generated by the $\hat{y}^{(i)}$. If the sequence does not cycle, she chooses another message m and reiterates the process. The first bit of s is $s_0 = 1$ since s must be odd. To find the next $(\mu - 1)$ bits $s_1, \ldots, s_{\mu-1}$, Carol asks to Alice to sign some $\hat{y}^{(i)}$ and successively evaluates

$$\hat{\sigma}_j := \frac{\widehat{\mathbb{S}}(\hat{y}^{(\mu-1-j)})}{\widehat{\mathbb{S}}(\hat{y}^{(\mu-1+\lambda-j \bmod \lambda)})} \bmod n \quad \text{for } j = 1, \ldots, \mu - 1. \tag{4}$$

If $\hat{\sigma}_j \equiv \hat{\sigma}_{j-1} \pmod{n}$ then $s_j = 0$; otherwise $s_j = 1$.

*We write $\hat{y}$ instead of y to indicate that the value is corrupted.

$K_1 :=$ `correct bits of` $\hat{y}^{(1)}$
$K_2 :=$ `correct bits of` $\hat{y}^{(2)}$
$k := 1$
`while` $(K_1 \neq K_2)$ `do`
    $K_1 := f(K_1)$; $K_2 := f(f(K_2))$; $k := k+1$
`od`
$K := K_1$
$K_1 :=$ `correct bits of` $\hat{y}^{(0)}$
$K_2 := K$
$\mu := 0$
`while` $(K_1 \neq K_2)$ `do`
    $K_1 := f(K_1)$; $K_2 := f(K_2)$; $\mu := \mu+1$
`od`
$K_1 := K$; $K_2 := f(K_1)$
$\lambda := 1$
`while` $(K_1 \neq K_2)$ `do`
    $K_2 := f(K_2)$; $\lambda := \lambda+1$
`od`

Figure 5: Kangaroos' method.

Proof. Assume that Carol already computed $\hat{\sigma}_1, \ldots, \hat{\sigma}_{k-1}$ and therefore knows $s_0, s_1, \ldots, s_{k-1}$. Then, she computes

$$\hat{\sigma}_k \equiv \frac{\widehat{\mathbb{S}}(\hat{y}^{(\mu-1-k)})}{\widehat{\mathbb{S}}(\hat{y}^{(\mu-1+\lambda-k \bmod \lambda)})} \equiv \prod_{i=0}^{t-1} \frac{\left(\hat{y}^{(\mu-1-k+i)}\right)^{s_i}}{\left(\hat{y}^{(\mu-1+\lambda-k \bmod \lambda+i)}\right)^{s_i}} \pmod{n}.$$

Suppose that $i > k$. Then letting $i = k+1+j$ with $j \geq 0$, we have

$$\hat{y}^{(\mu-1-k+i)} \equiv \hat{y}^{(\mu+j)} \pmod{n}$$

and

$$\begin{aligned} \hat{y}^{(\mu-1+\lambda-k \bmod \lambda+i)} &\equiv \hat{y}^{(\mu+\lambda+j+k-k \bmod \lambda)} \equiv \hat{y}^{(\mu+j+\lambda(1+\lfloor k/\lambda \rfloor))} \\ &\equiv \hat{y}^{(\mu+j)} \pmod{n}. \end{aligned}$$

So,

$$\hat{\sigma}_k \equiv \prod_{i=0}^{k} \frac{\left(\hat{y}^{(\mu-1-k+i)}\right)^{s_i}}{\left(\hat{y}^{(\mu-1+\lambda-k \bmod \lambda+i)}\right)^{s_i}} \equiv \hat{\sigma}_{k-1} \left(\frac{\hat{y}^{(\mu-1)}}{\hat{y}^{(\mu-1+\lambda)}}\right)^{s_k} \pmod{n}.$$

□

There is no general recipe to find the remaining bits of s. If the length μ of the tail is short, Carol may choose another random message m to find the μ first bits of s. The next bits have to be found by exhaustive search. However, this search can be speeded up by noting that the unknown bits have to fulfill some algebraic relations. For example, the bits $s_\mu, \ldots, s_{t-1}$ must satisfy the following relations:

$$\left(\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+i)}\right)^{\sum_{j=\mu}^{t-1} s_j} \equiv \frac{\prod_{i=0}^{\lambda-1} \widehat{\mathbb{S}}(\hat{y}^{(\mu+i)})}{\left(\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+i)}\right)^{\sum_{j=0}^{\mu-1} s_j}} \pmod{n} \tag{5}$$

and

$$\prod_{i=0}^{\lambda-1} \left(\frac{\hat{y}^{(\mu+i)}}{\hat{y}^{(\mu+(i+1) \bmod \lambda)}}\right)^{\sum_{\substack{0\le j\le t-1-\mu \\ j \bmod \lambda = i}} s_{j+\mu}} \equiv \frac{\widehat{\mathbb{S}}(\hat{y}^{(0)}) / \widehat{\mathbb{S}}(\hat{y}^{(1)})}{\prod_{j=0}^{\mu-1} \left(\frac{\hat{y}^{(j)}}{\hat{y}^{(j+1)}}\right)^{s_j}} \pmod{n}. \tag{6}$$

Proof.

$$\begin{aligned}
\prod_{i=0}^{\lambda-1} \widehat{\mathbb{S}}(\hat{y}^{(\mu+i)}) &\equiv \prod_{i=0}^{\lambda-1} \prod_{j=0}^{t-1} \left(\hat{y}^{(\mu+i+j)}\right)^{s_j} \equiv \prod_{i=0}^{\lambda-1} \prod_{j=0}^{t-1} \left(\hat{y}^{(\mu+(i+j) \bmod \lambda)}\right)^{s_j} \\
&\equiv \prod_{j=0}^{t-1} \left[\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+(i+j) \bmod \lambda)}\right]^{s_j} \equiv \prod_{j=0}^{t-1} \left[\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+i)}\right]^{s_j} \\
&\equiv \left(\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+i)}\right)^{\sum_{j=0}^{t-1} s_j} \\
&\equiv \left(\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+i)}\right)^{\sum_{j=0}^{\mu-1} s_j} \left(\prod_{i=0}^{\lambda-1} \hat{y}^{(\mu+i)}\right)^{\sum_{j=\mu}^{t-1} s_j} \pmod{n}.
\end{aligned}$$

$$\begin{aligned}
\frac{\widehat{\mathbb{S}}(\hat{y}^{(0)})}{\widehat{\mathbb{S}}(\hat{y}^{(1)})} &\equiv \prod_{j=0}^{\mu-1} \left(\frac{\hat{y}^{(j)}}{\hat{y}^{(j+1)}}\right)^{s_j} \prod_{j=\mu}^{t-1} \left(\frac{\hat{y}^{(j)}}{\hat{y}^{(j+1)}}\right)^{s_j} \\
&\equiv \prod_{j=0}^{\mu-1} \left(\frac{\hat{y}^{(j)}}{\hat{y}^{(j+1)}}\right)^{s_j} \prod_{j=0}^{t-1-\mu} \left(\frac{\hat{y}^{(\mu+j \bmod \lambda)}}{\hat{y}^{(\mu+(j+1) \bmod \lambda)}}\right)^{s_{j+\mu}} \\
&\equiv \prod_{j=0}^{\mu-1} \left(\frac{\hat{y}^{(j)}}{\hat{y}^{(j+1)}}\right)^{s_j} \prod_{i=0}^{\lambda-1} \left(\frac{\hat{y}^{(\mu+i)}}{\hat{y}^{(\mu+(i+1) \bmod \lambda)}}\right)^{\sum_{\substack{0\le j\le t-1-\mu \\ j \bmod \lambda = i}} s_{j+\mu}} \pmod{n}.
\end{aligned}$$

□

Note that if $\lambda = 2$, then Eq. (6) simplifies to

$$\left(\frac{\hat{y}^{(\mu)}}{\hat{y}^{(\mu+1)}}\right)^{\sum_{\substack{0\le j\le t-1-\mu \\ j\text{ even}}} s_{j+\mu}-\sum_{\substack{0\le j\le t-1-\mu \\ j\text{ odd}}} s_{j+\mu}} \equiv \frac{\widehat{\mathbb{S}}(\hat{y}^{(0)})/\,\widehat{\mathbb{S}}(\hat{y}^{(1)})}{\prod_{j=0}^{\mu-1}\left(\frac{\hat{y}^{(j)}}{\hat{y}^{(j+1)}}\right)^{s_j}} \pmod{n}. \tag{7}$$

Similar relations can be exhibited if λ is a power of 2.

2.3 Experimental results

The previous attack was implemented in order to check its effectiveness. For a 512-bit RSA-modulus, we observed that if the number of faulty bits is smaller than 20, then the length μ of the tail is generally long and all the bits of the secret exponent s can be recovered. The length λ of the cycle is also of some importance. The number of required faulty signatures actually depends on it. To find the μ first bits of s, we need to know $\mu + \lambda$ faulty signatures (see Eq. (4)).

Remark. Although quite efficient in practice, our attack is not fully optimized. It can be enhanced by searching collisions instead of cycles. This will be done in a future work.

3 Conclusions

The soundness of protocols can be verified by formal methods and thanks to the experience of the designer of good and bad practice. However this is not enough, flaws can be found in a well-designed protocol. Protocols must also be carefully implemented. Usually, programmers use some tricks in order to speed up the computations. In this paper, we illustrate this topic by showing how some secret information might be recovered if the right-to-left square-and-multiply algorithm was used to perform modular exponentiations. This proves once more that security and efficiency are two natural but conflicting goals in cryptography. Note that our attack does not apply to the left-to-right square-and-multiply algorithm. Therefore, the formal definition of correctness for cryptographic protocols must take into account the implementation (hardware and software).

The second warning of this paper is that the protocols must be faults resistant, or at least must provide for how to react in the case of faults. The correctness verification cannot always be achieved by doing calculations twice as proposed in [3]. Our cycling attack can pass through this test. The verification must be done by another (proved secure) way. For example, the Lenstra's attack (see Fig. 2) can be avoided as follows [14]. We use the same notations as in Section 1 (Introduction). Bob first chooses a (small) random number r relatively prime to n_B. Then he computes $m_{rp} = c^{d_B \bmod \phi(rp)} \bmod rp$ and $m_{rq} = c^{d_B \bmod \phi(rq)} \bmod rq$. If $m_{rp} \equiv m_{rq} \pmod{r}$, then the computations are assumed correct and $m = \mathrm{CRT}(m_{rp} \bmod p, m_{rq} \bmod q)$.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *1994 IEEE Symposium on Security and Privacy*, pages 122–136. IEEE Press, 1994.

[2] R. Anderson and R. Needham. Robustness principles for public key protocols. In D. Coppersmith, editor, *Advances in Cryptology – Crypto '95*, volume 963 of *Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag, 1995.

[3] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. on Computer Systems*, 8(1):18–36, February 1990.

[5] G. Davida. Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem. Technical Report TR-CS-82-2, Dept. of Electrical Engineering and Computer Science, University of Wisconsin, Milwaukee, USA, October 1982.

[6] A. Gillet, M. Joye, and J.-J. Quisquater. Cycling attacks against faulty hardware. Technical Report CG-1997/6, UCL Crypto Group, Louvain-la-Neuve, June 1997.

[7] M. Joye, A. K. Lenstra, and J.-J. Quisquater. Chinese remaindering in the presence of faults. Submitted to Journal of Cryptology.

[8] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[9] D. E. Knuth. *The art of computer programming: Volume 2/Seminumerical algorithms*. Addison-Wesley, 2nd edition, 1981.

[10] A. K. Lenstra. Memo on RSA signature generation in the presence of faults, September 1996.

[11] J. H. Moore. Protocol failures in cryptosystems. In G. Simmons, editor, *Contemporary Cryptology*, pages 541–558. IEEE Press, 1992.

[12] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[13] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[14] A. Shamir. How to check modular exponentiation. Presented at the rump session of Eurocrypt '97.