On-Board RSA Key Generation

An industrial experience

Marc Joye

technicolor

RSA Is Everywhere

- RSA = 95% of security products
  - Alternative technology: elliptic curves
- RSA comes in many standards

  Encryption PKCS #1 (RSA-OAEP), IEEE P1363a
  Signature PKCS #1 (PSS/PSS-R), ISO/IEC 9796 (RW), ANSI X9.31, NIST/FIPS PUB 186-2, ITU-T X.509
- RSA has been impacting smart-card technologies for 15+ years
  - Each and every chip manufacturer proposes its own cryptoprocessor(s) = specific hardware design(s)
  - Designing a cryptoprocessor = huge investments
    - financially
    - technologically (heavy devs, strong patents)
  - RSA performances are critical for all PK-enabled smart cards

My First Job

- Sep 1, 1999: I join Gemplus and receive my first assignment

Development of an on-board prime generation algorithm (6 months)

- Sep 21, 1999: The implementation is completed! Well almost. . .

technicolor

Off-Board/On-Board Key Generation

Off-board = keys generated in perso

- This is less secure for the end customer
- No dynamic control of key sizes, no re-generation

On-board key generation

- More secure for the end customer
- Re-generation on demand, dynamically-chosen sizes
- Applications can manage keys on their own
- Opens the way to key compression
  - e.g., 1024-bit RSA key $\mapsto$ 20 bytes

technicolor

①

Make It Functional

– or – how to generate RSA keys

RSA Key Generation

Main step (complicated...)

- On input $(\mathrm{random}, \ell, e)$, construct

$$q \leftarrow \texttt{GenPrime}(\mathrm{random}, \ell, e)$$

(Invoke this twice to get p, q)

Key derivation functions (easy)

- On input (e, p, q), compute
  - $N = p\,q$
  - $\begin{cases} d = e^{-1} \bmod (p-1)(q-1) & \text{(STD mode)} \\ d_p, d_q, i_q & \text{(CRT mode)} \end{cases}$

Classical Algorithms

Parameter: $\Pi = \prod_i p_i$
Output: a random ℓ-bit prime q

1. Randomly choose an ℓ-bit integer q
2. If $(\gcd(q, \Pi) \neq 1)$ then go to Step 1
3. If $(\mathsf{T}(q) = \texttt{false})$ then
   - $q \leftarrow q + \Pi$
   - Go to Step 3
4. Output q

- Usually, 10 first primes are used: $\Pi = 2 \cdot 3 \cdots 29$
  - H$[\ell, 10]$
- Naive generator corresponds to $\Pi = 2$

technicolor

Specification of GenPrime

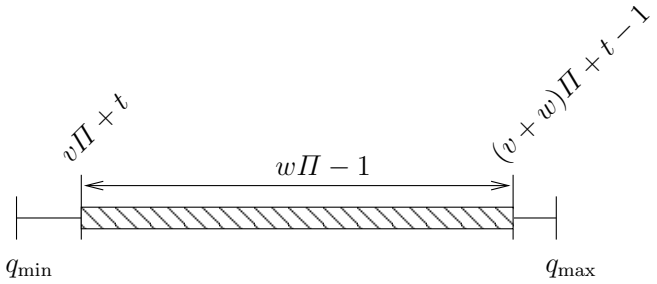
A prime number q generated by GenPrime in such that

1. q is an ℓ-bit number for a given bitsize ℓ
2. q belongs to $[q_{\min}, q_{\max}]$, e.g., $q_{\min} = \lceil 2^{\ell - 1/2} \rceil$ and $q_{\max} = 2^{\ell}$
3. $\gcd(q - 1, e) = 1$ where e is given

technicolor

10ème Anniversaire du Master Security, Cryptology and Coding of Information systems • Grenoble, Sept. 12, 2011

Choice of Parameters



- The prime candidates lie in

$$[v\Pi + t, (v+w)\Pi + t - 1] \subseteq [q_{\min}, q_{\max}]$$

- The prime candidates are automatically coprime to

$$\Pi = \prod p_i$$

technicolor

GenPrime

Parameters: Π, t, v, w and $a \in \mathbb{Z}_m^* \setminus \{1\}$
Output: a random prime $q \in [q_{\min}, q_{\max}]$

1. Compute $l \leftarrow v\Pi$ and $m \leftarrow w\Pi$
2. Choose $k \in_R \mathbb{Z}_m^*$
3. Set $q \leftarrow [(k - t) \bmod m] + t + l$
4. If $(\mathsf{T}(q) = \texttt{false})$ then
   - Set $k \leftarrow a \cdot k \pmod m$
   - Go to Step 3
5. Output q

$$\left.\begin{array}{l} q \bmod \Pi \equiv [k-t] + t + 0 \equiv k \pmod \Pi \\ k^{(\text{new})} = a \cdot k^{(\text{old})} \in \mathbb{Z}_m^* \implies k^{(\text{new})} \in \mathbb{Z}_\Pi^* \end{array}\right\} \implies \gcd(q, \Pi) = 1$$

technicolor

②

Make It Efficient

– or – how to [efficiently] generate RSA keys

Specification of `GenPrime` 2

A prime number q generated by `GenPrime` 2 in such that

1. q is an ℓ-bit number for a given bitsize ℓ
2. q belongs to $[q_{\min}, q_{\max}]$, e.g., $q_{\min} = \lceil 2^{\ell-1/2} \rceil$ and $q_{\max} = 2^{\ell}$
3. $\gcd(q-1, e) = 1$ where e is given

Further,

1. `GenPrime` 2 is "fast"
2. `GenPrime` 2 can accommodate a large value for Π
   $\Longrightarrow$ efficient generation of units
3. ℓ has a granularity of 1 bit, e.g., with $\ell \in [128, \ldots, 2048]$

GenPrime 2

Parameters: Π odd, $b_{\min}$, $b_{\max}$, v
Output: a random prime $q \in [q_{\min}, q_{\max}]$

1. Compute $I \leftarrow v\Pi$
2. Choose $k \in_R \mathbb{Z}_{\Pi}^*$
3. Choose $b \in_R \{b_{\min}, \ldots, b_{\max}\}$ and set $t \leftarrow b\,\Pi$
4. Set $q \leftarrow \begin{cases} k + t + I \\ (\Pi - k) + t + I \end{cases}$ (q is odd)
5. If $(\mathsf{T}(q) = \texttt{false})$ then
   - Set $k \leftarrow 2k \pmod{\Pi}$
   - Go to Step 4
6. Output q

Generation of Units (1/2)

Proposition

Let k, r be integers modulo m and assume $\gcd(r, k, m) = 1$. Then

$$k \leftarrow [k + r(1 - k^{\lambda(m)}) \bmod m] \in \mathbb{Z}_m^*$$

10ème Anniversaire du Master Security, Cryptology and Coding of Information systems • Grenoble, Sept. 12, 2011

technicolor

Generation of Units (2/2)

$$\gcd(r, k, m) = 1 \implies k \leftarrow [k + r(1 - k^{\lambda(m)}) \bmod m] \in \mathbb{Z}_m^*$$

Algorithm

1. Randomly choose $k \in [1, m[$
2. Set $U \leftarrow (1 - k^{\lambda(m)}) \bmod m$
3. If $(U \neq 0)$ then
   1. Choose a random $r \in [1, m[$
   2. Set $k \leftarrow k + rU \pmod m$ ["self-correctness"]
   3. Go to Step 2
4. Return $k \in \mathbb{Z}_m^*$

technicolor

RSA Primes (1/2)

An RSA prime q must satisfy $\gcd(e, q - 1) = 1$

Arbitrary public exponent e

- The test $\gcd(e, q - 1) = 1$ should be explicitly added

"Small" public exponent e

- Let $e = \prod_i e_i^{\nu_i}$
- If $e_i \mid \Pi$ for all i then our algorithms can be adapted such that the condition $\gcd(e, q - 1) = 1$ is automatically satisfied
  - This includes the popular choices $e = 3$ or $e = 17$

10ème Anniversaire du Master Security, Cryptology and Coding of Information systems • Grenoble, Sept. 12, 2011

technicolor

RSA Primes (2/2)

Exponent $e = \prod_i e_i^{\nu_i}$ and $e_i \mid \Pi$ for all i

1. Let $\alpha \in \mathbb{Z}_m^*$ be such that
   - $\mathrm{order}(\alpha \bmod e_i, e_i) = e_i - 1$ for each $e_i \mid e$
2. Define $a = \alpha^2 \bmod m$
3. Set $k^{(0)} \equiv \alpha \pmod{e^+}$ where $e^+ = \prod_i e_i$

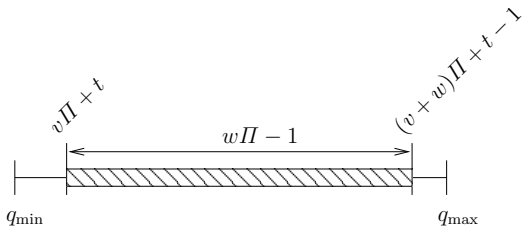
Hence

$$q \equiv k \equiv \alpha^{2j+1} \pmod{e^+} \text{ for some } j \text{ (because } e^+ \mid \Pi)$$

$$\begin{aligned} &\Longrightarrow q \not\equiv 1 \pmod{e^+} && \text{(since } \alpha \text{ is of even order modulo } e^+) \\ &\Longrightarrow \gcd(q-1, e_i) = 1 && \text{for all } e_i \\ &\Longrightarrow \gcd(q-1, e) = 1 \end{aligned}$$

technicolor

Length Extendability



- Parameters of `GenPrime` are
  - (Π, t, v, w)
  - $\lambda(m)$ with $m = w\Pi$
- Heavily depend $q_{\min} = \lceil 2^{\ell_0 - 1/2} \rceil$ and $q_{\max} = 2^{\ell_0}$

Scalability

Our algorithms allow to use the parameters sized for ℓ_0 to generate primes of bitsize $\ell \geqslant \ell_0$

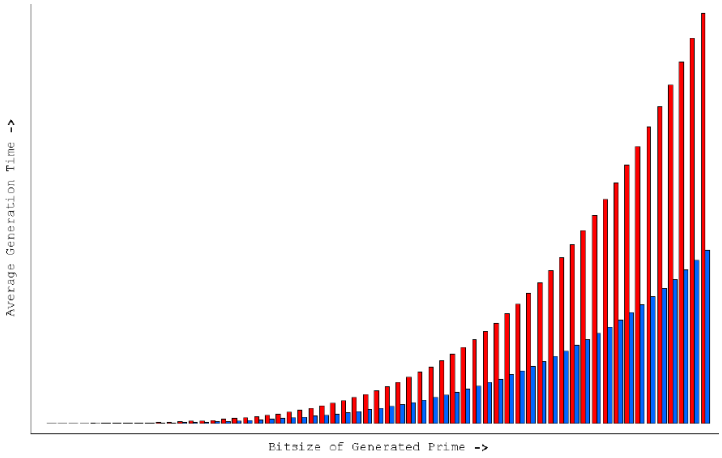
technicolor

10ème Anniversaire du Master Security, Cryptology and Coding of Information systems • Grenoble, Sept. 12, 2011

Performance Analysis

Average number of primality tests for generating q

Bitsize ℓ	256	384	512	768	1024
H[ℓ, 10]	28.03	42.04	56.05	84.08	112.1
GenPrime[ℓ]	18.72	26.12	33.29	46.90	59.98



technicolor

③

Make It Compliant

– or – how to generate [standard] RSA keys

DSA Primes

A DSA prime q is such that $q = 1 + q'r$ for a 160-bit prime q'

Goal: Constructively generate DSA prime candidate q coprime to Π

$$q \not\equiv 0 \pmod{p_i} \iff r \not\equiv -\frac{1}{q'} \pmod{p_i} \quad \text{for all } p_i \mid \Pi$$

Algorithm

1. Set $$r \leftarrow -\frac{1}{q'} + k \pmod{m} \quad \text{for some } k \in_R \mathbb{Z}_m^*$$
2. Set $q \leftarrow 1 + q'r$
3. If $(\mathrm{T}(q) = \texttt{false})$ then
   - Set $k \leftarrow a \cdot k \pmod{m}$
   - Go to Step 1

technicolor

X9.31 Primes

A X9.31 prime q is such that $q - 1 = r_1\, u$ and $q + 1 = r_2\, s$ for "large" primes u and $s \implies r_1 \equiv -\frac{2}{u} \pmod{s}$

Hence

$$q = 1 + u\left(-\frac{2}{u} \bmod s + r\, s\right) = \mathbb{1} + q'r$$

with $\mathbb{1} = 1 + u(-2/u \bmod s)$ and $q' = u\, s$

Algorithm

As for DSA primes, set

$$r \leftarrow -\frac{\mathbb{1}}{q'} + k \pmod{m} \quad \text{for some } k \in_R \mathbb{Z}_m^*$$

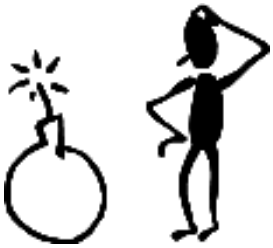
technicolor

10ème Anniversaire du Master Security, Cryptology and Coding of Information systems • Grenoble, Sept. 12, 2011

④

Make It Secure

– or – how to generate [securely] RSA keys



Square-and-Multiply Algorithm (1/3)

- Square-and-multiply algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

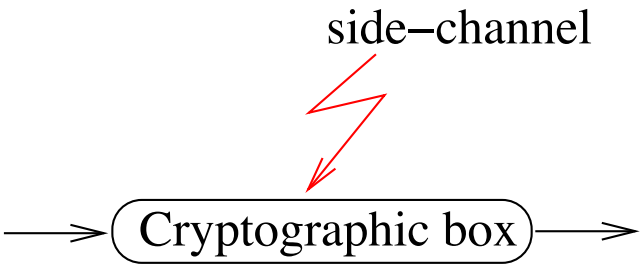
---

1. $R_0 \leftarrow 1$
2. For $i = k - 1$ downto 0 do
   - $R_0 \leftarrow {R_0}^2 \pmod N$
   - If $(d_i = 1)$ then $R_0 \leftarrow R_0\, \dot{m} \pmod N$
3. Return R_0

- $1 + 1$ temporary variables (R_0 and $\dot{m}$)
- . . . subject to SPA-type attacks

Square-and-Multiply Algorithm (2/3)

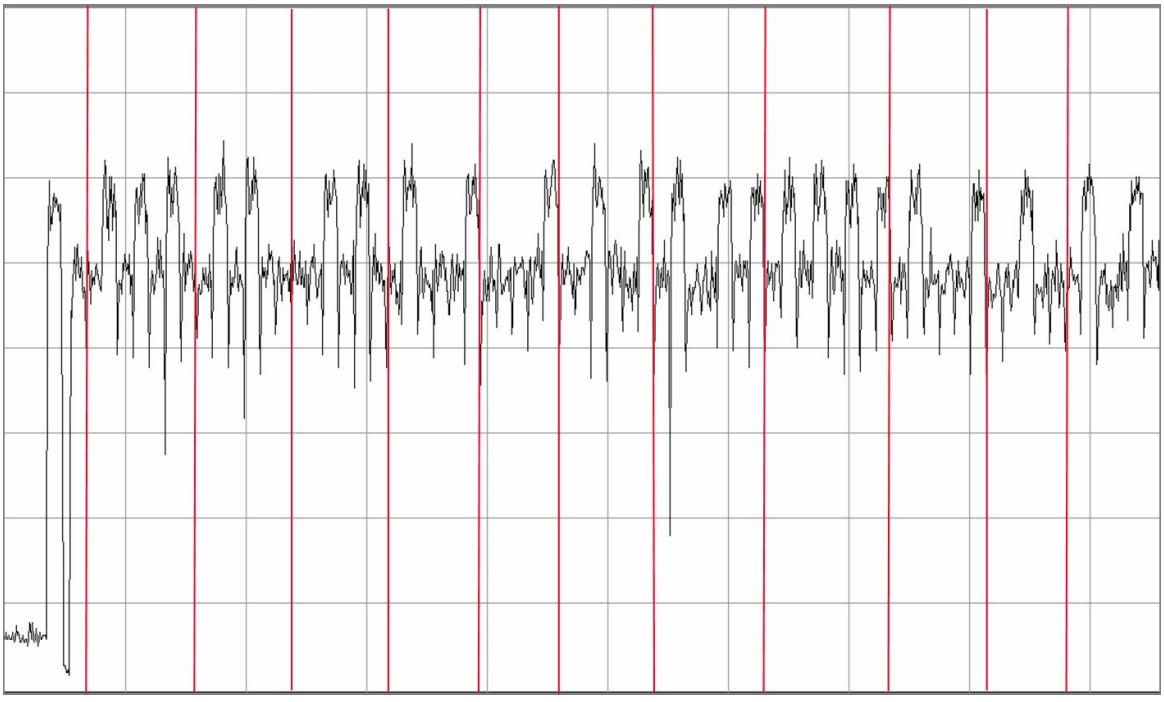
- Simple side-channel analysis



  - side-channel = timing, power consumption, . . .

technicolor

Square-and-Multiply Algorithm (3/3)



Key: $d = \texttt{2E C6 91 5B FE 4A} \ldots$

technicolor

Square-and-Multiply-*Always* Algorithm

- Square-and-multiply-*always* algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

---

1. $R_0 \leftarrow 1$; $R_1 \leftarrow 1$
2. For $i = k - 1$ downto 0 do
   - $R_0 \leftarrow {R_0}^2 \pmod N$
   - $b \leftarrow 1 - d_i$; $R_b \leftarrow R_b \, \dot{m} \pmod N$
3. Return R_0

- when $b = 1$ (i.e., $d_i = 0$), there is a dummy multiplication
- the power trace now appears as a regular succession of squares and multiplies
- $2 + 1$ temporary variables (R_0, R_1 and $\dot{m}$)

Montgomery Powering Ladder

- Montgomery exponentiation algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

---

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$
2. For $i = k - 1$ downto 0 do
   - $b \leftarrow 1 - d_i$; $R_b \leftarrow R_0 \, R_1 \pmod N$
   - $R_{d_i} \leftarrow {R_{d_i}}^2 \pmod N$
3. Return R_0

- behaves regularly without dummy operations
- only 2 temporary variables (R_0, R_1)

Summary

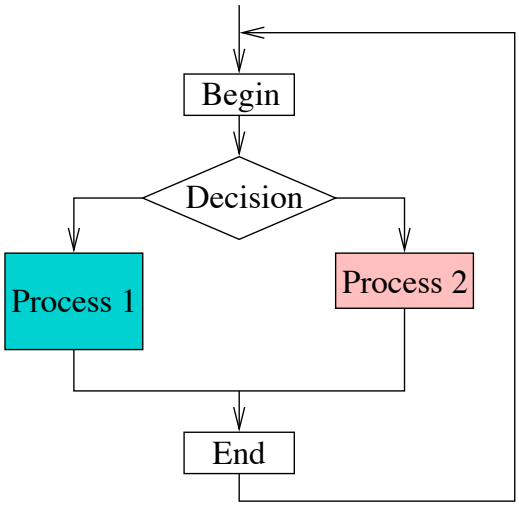
- Comparison

Algorithm	Temp. var.	# mult.
Square-and-multiply	$1+1$	$k+k/2$
Square-and-multiply-*always*	$2+1$	$k+k$
Montgomery ladder	2	$k+k$

- Side-channel atomicity
  - converts any crypto-algorithm into a protected algorithm with (virtually) no penalty

General Principle

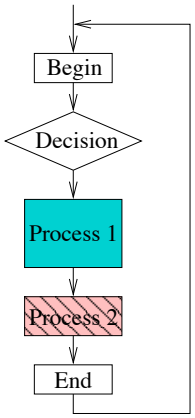
- Example of a crypto-algorithm



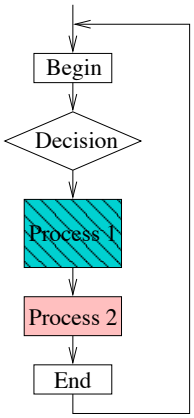
- Side-channel information
  - timing, power consumption, etc. . .

General Principle (1/3)

- Straightforward solution



. . . for executing Process 1



. . . for executing Process 2

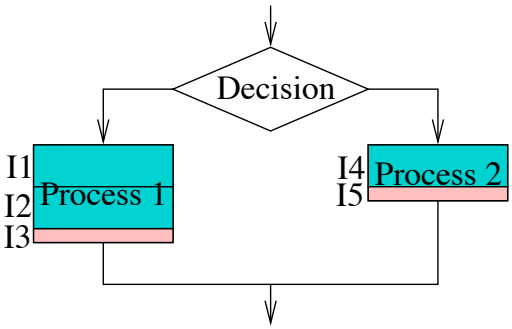
provided that a fake execution is indistinguishable from a true execution!

technicolor

General Principle (2/3)

Side-channel atomicity

Refinement of the straightforward solution so that the running time is not (too much) penalized



Process 1 = $\mathrm{I1}||\mathrm{I2}||\mathrm{I3}$ and Process 2 = $\mathrm{I4}||\mathrm{I5}$

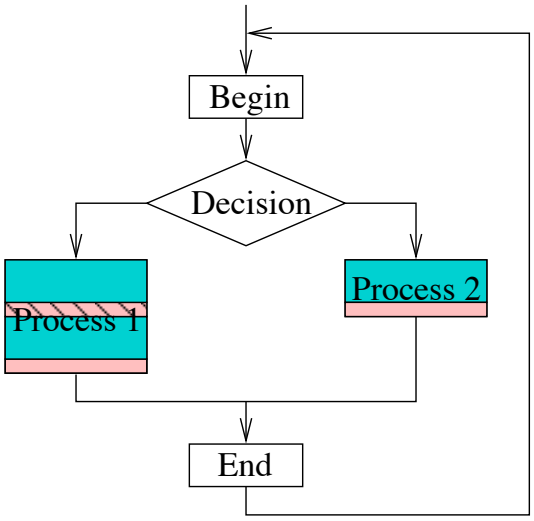
Common atomic block

 : $\mathrm{I1}||\mathrm{I3}^{(\mathrm{fake})} \sim \mathrm{I2}||\mathrm{I3} \sim \mathrm{I4}||\mathrm{I5}$

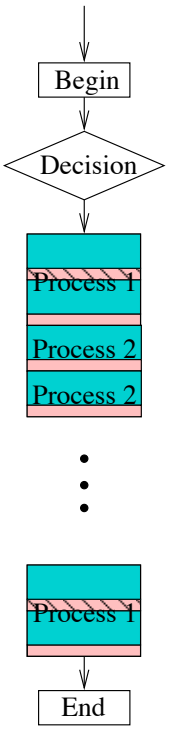
technicolor

General Principle (3/3)

- The whole crypto-algorithm



with chained blocks →

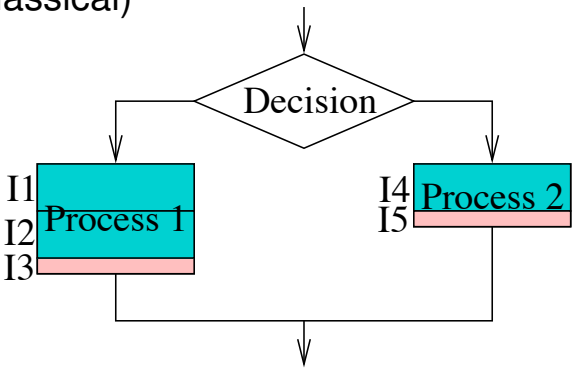


technicolor

Atomic Square-and-Multiply (1/3)

- Application of the 'General Principle'
  - square-and-multiply algorithm (classical)

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$
2. While $(i \geqslant 0)$ do
   - $R_0 \leftarrow {R_0}^2 \pmod N$
   - If $(d_i = 1)$ then $R_0 \leftarrow R_0 R_1 \pmod N$
   - $i \leftarrow i-1$
3. Return R_0



Assumptions

- $[R_0 \leftarrow R_0 R_0 \pmod N] \sim [R_0 \leftarrow R_0 R_1 \pmod N]$
- $[i \leftarrow i-1] \sim [i \leftarrow i-0]$

technicolor

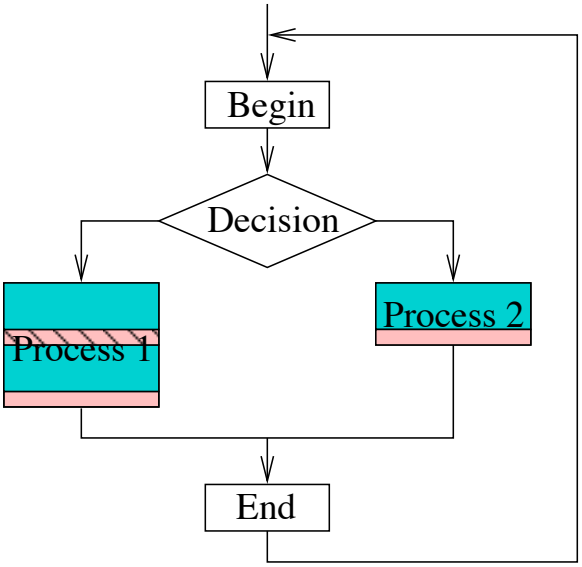
Atomic Square-and-Multiply (2/3)

1 $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$

2 While $(i \geqslant 0)$ do
Case
$(d_i = 1)$: /* Process 1 */
- $R_0 \leftarrow R_0\, R_0 \pmod N$
- $i \leftarrow i - 0$
- $R_0 \leftarrow R_0\, R_1 \pmod N$
- $i \leftarrow i - 1$

$(d_i = 0)$: /* Process 2 */
- $R_0 \leftarrow R_0\, R_0 \pmod N$
- $i \leftarrow i - 1$

3 Return R_0



technicolor

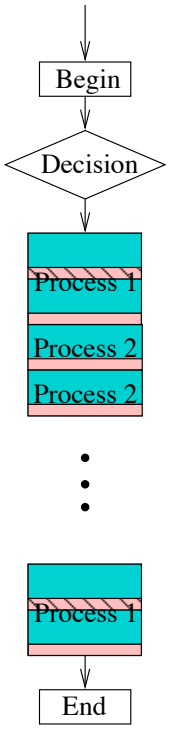
Atomic Square-and-Multiply (3/3)

1 $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$

2 While $(i \geqslant 0)$ do
Case
$(d_i = 1)$: /* Process 1 */
- $R_0 \leftarrow R_0\, R_0 \pmod N$
- $i \leftarrow i - 0$
- $R_0 \leftarrow R_0\, R_1 \pmod N$
- $i \leftarrow i - 1$

$(d_i = 0)$: /* Process 2 */
- $R_0 \leftarrow R_0\, R_0 \pmod N$
- $i \leftarrow i - 1$

3 Return R_0



Chaining the blocks

technicolor

Chaining the Blocks (1/4)

Methodology

1. Each process is divided into common atomic blocks
2. Each block inside a process receives a number r (in chronological order)
3. A bit s is used to keep track whether there remain blocks to be executed in the current process

The trick:

$$r \leftarrow (\neg s) \cdot (r+1) + s \cdot f(\text{input values})$$

- $s = 0 \implies r \leftarrow r + 1$
- $s = 1 \implies r \leftarrow f(\text{input values})$

technicolor

Chaining the Blocks (2/4)

- Square-and-multiply algorithm

		r	s
$(d_i = 1)$	$R_0 \leftarrow R_0\, R_0 \pmod N;\ i \leftarrow i - 0$	0	0
	$R_0 \leftarrow R_0\, R_1 \pmod N;\ i \leftarrow i - 1$	1	1
$(d_i = 0)$	$R_0 \leftarrow R_0\, R_0 \pmod N;\ i \leftarrow i - 1$	2	1

- We can choose for the common atomic block ()

$$R_0 \leftarrow R_0\, R_{(r \bmod 2)} \pmod N;\ i \leftarrow i - s$$

and for the update

$$\begin{cases} r = (\neg s) \cdot (r+1) + s \cdot f(d_i) & \text{with } f(d_i) = 2 \cdot (\neg d_i) \\ s = (r \bmod 2) + (r \operatorname{div} 2) \end{cases}$$

technicolor

Chaining the Blocks (3/4)

- Resulting algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$; $s \leftarrow 1$
2. While $(i \geqslant 0)$ do
   - $r \leftarrow (\neg s) \cdot (r+1) + s \cdot 2(\neg d_i)$
   - $s \leftarrow (r \bmod 2) + (r \operatorname{div} 2)$
   - $R_0 \leftarrow R_0 \cdot R_{(r \bmod 2)}$; $i \leftarrow i - s$
3. Return R_0

- after simplification...

Chaining the Blocks (4/4)

- Atomic square-and-multiply algorithm

Input: $\dot{m}, d = (d_{k-1}, \ldots, d_0)_2, N$
Output: $S = \dot{m}^d \bmod N$

1. $R_0 \leftarrow 1$; $R_1 \leftarrow \dot{m}$; $i \leftarrow k-1$; $b \leftarrow 0$
2. While $(i \geqslant 0)$ do
   - $R_0 \leftarrow R_0 R_b \pmod N$
   - $b \leftarrow b \oplus d_i$; $i \leftarrow i - \neg b$
3. Return R_0

- behaves regularly without dummy operations
- only 2 temporary variables (R_0 and R_1)

⑤

Make It General

– or – how to generate [any] RSA keys

RSA Key Generation

Main step (complicated. . .)

- On input $(\mathrm{random}, \ell, e)$, construct

$$q \leftarrow \texttt{GenPrime}(\mathrm{random}, \ell, e)$$

(Invoke this twice to get p, q)

Key derivation functions (easy)

- On input (e, p, q), compute
  - $N = p\,q$
  - $\begin{cases} d = e^{-1} \bmod (p-1)(q-1) & \text{(STD mode)} \\ d_p, d_q, i_q & \text{(CRT mode)} \end{cases}$

Computing Private Exponent d (or d_p, d_q)

Arazi's inversion formula

Let e and f be two positive integers. If $\gcd(e, f) = 1$ then

$$d = e^{-1} \bmod f = \frac{1 + f(-f^{-1} \bmod e)}{e}$$

- Application

  STD mode $f = \phi(N)$

  CRT mode $f = p - 1$ (and $q - 1$)

- Assumption: e is prime

$$\Longrightarrow f^{-1} \equiv f^{e-2} \pmod{e}$$

$$\Longrightarrow d = \frac{1 + f(-f^{e-2} \bmod e)}{e}$$

technicolor

Computing d: General Case (1/3)

$$d = \frac{1 + f(-f^{\lambda(e)-1} \bmod e)}{e}$$

- Different scenarios
  - e is provided along with $\lambda(e)$
  - e is provided but is known to be prime
  - nothing is known about e
- First attempt: assume e is smooth (or prime)
  1. Try
  $$\widehat{\lambda} = e(e-1)\Pi\,\lambda(\Pi)$$
  as a (candidate) multiple for $\lambda(e)$
  2. Compute $\widehat{d} = \frac{1 + f(-f^{\widehat{\lambda}-1} \bmod e)}{e}$
  3. Check that $e\widehat{d} \equiv 1 \pmod{f}$, and if so, return $\widehat{d}$

technicolor

Computing d: General Case (2/3)

A simple observation

$$e^{-1} \equiv (e + Cf)^{-1} \pmod{f}$$

- Choose C such that $\widehat{e} = e + Cf$ is prime and apply Arazi's formula
- Choose C such that $\gcd(\widehat{e}, \Pi) = 1$ with $\widehat{e} = e + Cf$
  - e.g.,
    $$C = \left[(a - e) f^{\lambda(\Pi)-1}\right] \bmod \Pi$$
    for some $a \in \mathbb{Z}_{\Pi}^{*}$

Computing d: General Case (3/3)

Parameters: e, f, Π, and $k \in \mathbb{Z}_m^* \setminus \{1\}$
Output: $d = e^{-1} \bmod f$

1. Compute $U \leftarrow f^{\lambda(\Pi)-1} \bmod \Pi$
2. Set $C \leftarrow \left[(a - e)\right] f^{\lambda(\Pi)-1} \bmod \Pi$ and $\widehat{e} \leftarrow e + Cf$
3. If $(\mathsf{T}(\widehat{e}) = \texttt{false})$ then
   - Set $a \leftarrow a \cdot k \pmod{\Pi}$
   - Go to Step 2
4. Compute $F \leftarrow -f^{\widehat{e}-2} \bmod \widehat{e}$
5. Output $d = (Ff + 1)/e$

Conclusion

– or – how RSA keys were generated

My First Job

- Sep 1, 1999: I join Gemplus and receive my first assignment

Development of an on-board prime generation algorithm (6 months)

- Sep 21, 1999: The implementation is completed! Well almost. . .
- June (?), 2000: The PK library was successfully integrated and validated!

Development of Security Software

Security requirements

Validation of design

Code review of critical security elements

Testing/Integration

First security test on functional software

Validation

Second security test on final software

Deployment: Collect vulnerabilities

technicolor

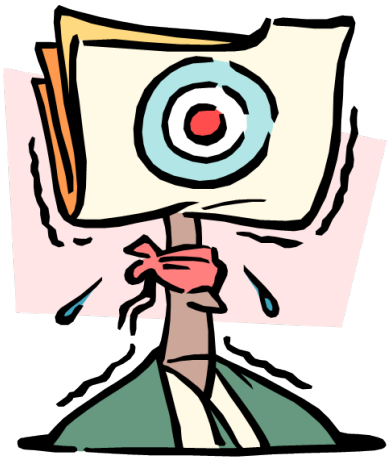
Research in Industry

- Industry may be an option if you like [applied] research
  - it gives naturally problems to work on
  - your ideas may be implemented in real products

technicolor

10ème Anniversaire du Master Security, Cryptology and Coding of Information systems • Grenoble, Sept. 12, 2011

Comments/Questions?



http://joye.site88.net/

technicolor