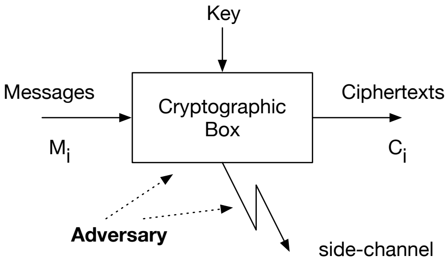
Timing Attacks and Countermeasures



Timing Attacks

- Processing time depends on the value of the secret key bit
- It leaks information about it
- There are ways to measure it
- Timing attack conditions
  - The processing should be monitored
  - Processing durations need to be recorded
  - Some basic computational and statistical tools are needed
  - Knowledge of the implementation will be required

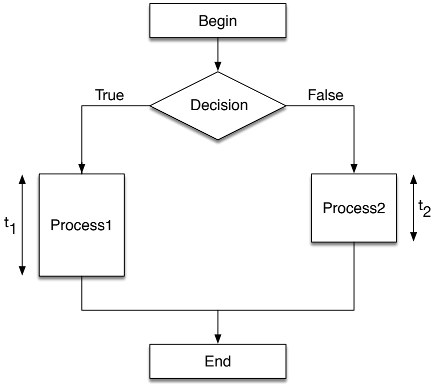
Timing Attacks

The code starts unconditionally

The test is based on secret bit

Depending on the Boolean condition the process may be long (t_1) or short (t_2)

The code continues unconditionally



Timing Attacks

- The term "Timing Attack" was introduced by Paul Kocher in 1996
- First practical attacks in Crypto 1997 Conference
- Applicable to RSA and in fact all cryptosystems
  - Basic mathematical operations
  - Modular exponentiation
  - Cryptographic algorithms
- Knowledge and variability of messages are needed
- Time measurements must be accurate to within few clock cycles

Attacking RSA Algorithm

- The standard RSA exponentiation $s = m^d \pmod{n}$
- The Montgomery method for modular multiplication
- Timing variations in Montgomery due to Subtraction Step
- The binary method of exponentiation yields bits of d

The Binary Method of Exponentiation

- Input: m, d, n
- m: message which is k bits
- (d, n): the RSA private key, k bits each
- Output: $s = m^d \pmod{n}$
- m: signature which is k bits

Input: m, $d = (d_{k-1}, \ldots, d_0)_2$, n
Output: $s = m^d \pmod{n}$
1. $s \leftarrow 1$
2. For $i = k - 1$ downto 0
$s \leftarrow s \cdot s \pmod{n}$
If $d_i = 1$ then $s \leftarrow s \cdot m \pmod{n}$
3. Return s

The Montgomery Modular Multiplication

- The Montgomery modular multiplication MonPro is a special, high-speed modular multiplication algorithm
- The function $\text{MonPro}(a, b)$ computes $a \cdot b \cdot r^{-1} \pmod{n}$
- Interestingly the algorithm does not need $r^{-1} \pmod{n}$
- However, it requires another quantity n' which is related to it
- It is significantly faster than Multiply-and-Reduce algorithm

Classical Montgomery Algorithm

- Peter Montgomery introduced his original algorithm in 1985
- It produces a result in the range $[0, 2n)$
- A subtraction may be required to fully reduce mod n

function MonPro(a, b)
Input: a, b, n, n'
Output: $u = a \cdot b \cdot r^{-1} \bmod n$
1: $t \leftarrow a \cdot b$
2: $m \leftarrow t \cdot n' \pmod{r}$
3: $u \leftarrow (t + m \cdot n)/r$
4: **if** $u \geq n$ **then** $u \leftarrow u - n$
5: **return** u

The Montgomery Modular Multiplication

- Multiply step for bit d_i
- If $d[i] = 1$ then $s = \text{MonPro}(s, m)$
- Step 1+2+3: The multiply-add steps of Montgomery multiplication
- Step 4: If the result is larger than n, a subtraction by n

Attacking RSA Algorithm

- Assume, we have obtain L message and signature pairs and their timings (m_j, s_j, t_j) such that the computation of $s_j = m_j^d \pmod{n}$ requires t_j seconds
- The private key is unknown, and we are trying to determine it
- Now assume, higher $(i-1)$ bits of the exponent d are discovered
- That is, we know $d[k-1], d[k-2], \ldots, d[k-(i-1)]$
- Knowing the message m_j, we can compute the **intermediate value of the signature** s_j^* after the square operation for index $(k-i)$

Attacking RSA Algorithm

- We can then determine whether the Montgomery multiplication operation $\text{MonPro}(s_j^*, m_j)$ will cause a subtraction
- However, we do not know the value of the bit $d[k-i]$
- If $d[k-i] = 0$, there will not be a Montgomery multiplication (and thus no subtraction either)
- if $d[k-i] = 1$, there will be a Montgomery multiplication and we have determined whether there will be a subtraction or not

Description of the Attack

- Let $\mathcal{S}$ the set of messages: $\mathcal{S} = \{m_1, m_2, \ldots, m_L\}$
- Let $\mathcal{T}$ the set of timings: $\mathcal{T} = \{t_1, t_2, \ldots, t_L\}$
- Assume $d[k-i] = 1$
- Partition $\mathcal{S}$ into two disjoint subsets: $\mathcal{S}_0$ and $\mathcal{S}_1$ such that
- $\mathcal{S}_0 = \{m_j \ : \ \text{MonPro}(s_j^*, m_j) \text{ does not have subtraction}\}$
- $\mathcal{S}_1 = \{m_j \ : \ \text{MonPro}(s_j^*, m_j) \text{ has subtraction}\}$

- Compute the mean time $\overline{\mathcal{T}}_0$ of the messages in $\mathcal{S}_0$
- Compute the mean time $\overline{\mathcal{T}}_1$ of the messages in $\mathcal{S}_1$

Description of the Attack

- Case $d[k-i] = 0$
  Global times for sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are **statistically indistinguishable** since the split is based on a multiplication which does not occur

- Case $d[k-i] = 1$
  Global times for sets $\mathcal{S}_0$ and $\mathcal{S}_1$ show a statistical difference to the optional multiplication since it does occur

- Time measurements validate or invalidate the assumption:

  If $\overline{\mathcal{T}}_0 - \overline{\mathcal{T}}_1 \gg 0$, the assumption is valid, that is $d[k-i] = 1$

  If $\overline{\mathcal{T}}_0 - \overline{\mathcal{T}}_1 \approx 0$, the assumption is wrong, that is $d[k-i] = 0$

Conclusions

- For 128 bits, the attack recovers 2 bits/sec for $L = 10,000$
- For 512 bits, the attack recovers 1 bits/sec for $L = 100,000$
- Together with other side-channel attacks they become more efficient
- It works against computers, servers, not just smart cards
- A real threat for many devices and computers

Countermeasures

- A basic countermeasure would be to create constant-time processing
- Blinding (whitening or randomization) approaches also work