System Security fiche

Pierre Colson

January 2023

Contents

Markdown version on *github*
Compiled using *pandoc* and *`gpdf` script*

Side Channels & Tempest

- **Compromising Emanations**: Physical signals related to digital activity; break the assumption of higher-level abstractions; Root cause of many attacks
- **Tempest**: Passive leakage of plaintext information (E.g., video on screen)
  - Video Signal
    - Signal in wire/connector/etc. not well shielded
    - Current in wires generates EM waves
    - Modulated with the pixel values
- **Soft Tempest**: Active version of *Tempest*; leakage used to exfiltrate data
  - Vide Signal
    - Signal in wire/connector/etc. not well shielded
    - Current in wires generates EM waves
    - Modulated witht the pixel values
    - Use to transmit data
    - Or to add noise Tempest leakage
    - Possible with many other sources of leakage (e.g. memory access)
- **Side Channels**: Use leakage to attack cryptographic implementation (Only in proximity, with few exceptions)
- Type of *pysical leakage*: Execution time, Power, Magnetic and electromagnetic, optical, thermal, acoustic and vibrational, reflection of injected signals
- Type of *attack*: Passively recover plaintext, Actively exfiltrate data; attack cyrptographic implementation
- **Symmetric encryption** for *confidentiality*
  - *Stream cypher*
    - Process a message bit by bit (byte by byte)
    - $KeyStream = PseudoRandomBitStreamGenerator(seed)$
    - $CyphertextStream = KeyStream + PlaintextStream$
  - *Block cypher*
    - Process a message block by block (EAS, DES)
    - Plaintext might need *padding*
    - $Plaintext + Key = BlockCipher$
    - All block cipher leads to the cipher text
    - There is different way to concatenate block cipher
    - **Electronic CodeBlock** (ECB, *insecure*) $C_1 = P_1 \bigoplus K$, $C_2 = P_2 \bigoplus K$, $\ldots$

    - **Cipher Block Chaining** (CBC) $C_1 = (P_1 \bigoplus IV) \bigoplus K$, $C_2 = (P_2 \bigoplus C_1) \bigoplus K$, ...
- **Symmetric crypto for authentication**
  - **Message Authentication Code (MAC)**
  - Shared key between A and B
  - A sends to B message $M + MAC$ where $MAC = MACFunction(M, K)$
- **Asymmetric crypto for confidentiality**
  - A and B exchange a key pair via a trusted channel
  - A wants to send M to B, she sends: $C = \mathit{Encryption}(Pu_B, M)$ where Pu_B is B's public key
  - B decrypts the message as follow: $M' = \mathit{Decryption}(Pr_B, C)$ where Pr_B is B's private key
- **Asymmetric crypto for authentication**
  - A send to B message M and Signature S, where $S = \mathit{Encryption}(Pr_A, M)$ where (A's private key)
  - B verifies the signature by checking: $\mathit{Decryption}(Pu_A, S) = M$ where (Pu_A is A's public key)
- Combine the best of two: Asymmetric key exchange + Symmetric encryption
- **Security** of cryptographic algoritms
  - We *model* system and possible attackers
  - Security properties are valid under certain *assumptions*
- Side Channel concrete example
  - **Timing**: Measure execution time
    - Classic timign attack against RSA
    - Remote attack are possible
    - Modern example of remote attack onc cryptocurrencies
  - **Power** Measure some physical quantity influenced by execution
    - Simple Power Analysis (SPA)
    - Differential Power Analysis (DPA)
    - Correlation Power Analysis (CPA)
- Reminder on **RSA**
  - *Key generation*
    1. Chose numbers p, q such that p and q are prime and $p \neq q$
    2. Compute $n = pq$
    3. Compute $\Phi(n) = \Phi(p-1)\Phi(q-1)$
    4. Chose e such that e and $\Phi(n)$ are relative prime and $1 < e < \Phi(n)$
    5. Compute d as such that $de \mod \Phi(n) = 1$
    6. Public key $PU = \{e, n\}$
    7. Private key $PR = \{d, n\}$
  - *Encryption*
    - Plaintext $m < n$
    - Ciphertext $C = m^e modn$
  - *Decryption*
    - Ciphertext C
    - Plaintext $m = C^d modn$
  - *Signature*
    - Plaintext $m < n$
    - Signature $s = m^d modn$
  - The *security* of RSA is based on two hard problems
    - The RSA problem, i.e., computing the e^{th} root of m modulo n from $C = m^e \mod n$
    - FActoring large numbers into smaller primes
- Exponentiation is implemented using *Square and multiply*
  - Problem 1
    - Key dependant branching
    - Execution time depends on the key d, if bit i of d is 0 is will be faster than if bit i of d is 1
  - Problem 2
    - Montgomery used for modular multiplication because it is more efficient
    - Montgomery execution time T_{mont} depends on the plaintext m; there is a reduction step done only if necessary

- **Countermeasures**
- **Constant time**
  - Relatively easy for specific cases
    - E.g., modular multiplication without conditional reduction
  - Generic protection is hard
    - Identify and elininate all dependencies of time with plaintext and key
    - Can have performance issues
- **What if we artificially add noise**
  - An attacker jus tneed more measurements to dig the signal out of the noise
- **Masking**: Can we make it impossible fot the attacker to guess
  - Mask with random number C different for each message:
    - $md \mod n \rightarrow [(m.X)d \mod n].[(X^{-1})d \mod n] \mod n$
  - Intuitively, given m and d_i the attacker cannot guess slow/fast any more
- A **logic gate**
  - Electronic component that implements a logic operator (not, and, nand, or, xor)
  - Stateless (Combinatorial)
  - Together with memory elements it is used to implement finit state machines
- *MOS transistor*: electronic switch
- Logical gate can be implemented with MOS
- **Data dependency**: There are physical phenomena that create a data dependency between logic values and their transitions and the power consumption of the circuit
- **Measure**
  - We can measure the power consumption and observe these phenomena
  - Signals are small, many measurements and statistical analysis are often needed
- **Model**: we know how it works: given some logic data manipulated by the software/hardware, we can predict the corresponding power consumption
- **Countermeasures**
  - *Problem*: There is a data dependency (of some order) between plaintext,key and the power
  - Add noise
    - Desyncronize ther traces
    - Inject random noise
    - Defeated with better signal processing and more measurements
  - Try to balance the hardware
    - Filtering shielding (Filtering is not perfect, expensive, can be tempered)
    - Make a processor where every instruction/operands consumes the same power (Not easy and expensive)
  - N^{th} order masking
    - Multiply each data with a random variable
    - This algorithmically breaks the dependecy making it impossible to guess the intermediate value
- **EM side channel**
  - Currents flowing in cables produce EM signals
  - Clock might act as a carrier
  - Emissions from localized areas, are not all overall power consumption
- **Sound side channel**
  - Currents in certain capacitors make them vibrate and produce sounds
- We don't always need a physical access
- **Tamper resistant** systems take the bank vault approach
  - Prevention of break in
- **Tamper responding** systems use the burglar alarm approach
  - Real-time detection of intrusion and prevention of access to sensitive data
- **Tamper evident** system are designed to ensure that is a break-in occurs, evidence of the break in is left behind
  - Detection of intrusion