Physical Attacks and Countermeasures

vincent.immler@oregonstate.edu

CS579/ECE599, January 12, 2023

fault torlance

SISD MISD
SIMD MIMD

multiple system

Introduction to Cryptography (Engineer's Perspective)

- Note: this is only intended as brief review of concepts that matter
- Classical scenario: Alice and Bob are honest parties
- The communication link between them is not secure
- They want to communicate *securely*
  - By using the same secret key: symmetric cryptography
  - By using public/private keys: asymmetric cryptography
- Eve can eavesdrop the communication
- Most security principles have been formulated with this in mind

Security Principles of Cryptography

- Kerckhoff's principle (1883): "*the security of a cryptosystem must lie in the choice of its keys only; everything else (including the algorithm itself) should be considered public knowledge.*"
  - It is easier to keep/change a secret key than a secret algorithm
  - Asymmetric schemes and communication with many parties are possible
  - Public scrutiny allows to inspect security and avoid backdoors
- "*A cryptographic scheme for a given task is secure, if no adversary of a specified power can achieve a specified break*" [Katz and Lindell]
- Issues of above definitions within the context of physical security
  - 'Cryptosystem' or 'scheme' clearly have an algorithm or protocol in mind
  - What about cryptographic devices that implement these?
  - What about non-cryptographic security objectives?
- The latter is more in line with nowadays security certifications (Common Criteria)
- Breaking a chip or cryptographic device means violating its security objective(s)

Security Principles of Modern Cryptography

- Cryptographic security principle: computational complexity
  - Most widely used; complexity expressed in terms of bits
  - Exhaustive key search (brute force) of key is not feasible
  - Symmetric ciphers typically require $\geq 80\,\mathrm{bit}$ for security
  - Asymmetric ciphers rely on mathematical trapdoor function
  - Assumption: inverting the trapdoor is difficult (no good algorithms known)
  - Security level depends on mathematical problem; risk of quantum computers
- Cryptographic security principle: indistinguishability
  - Attacker is given output of valid cryptographic protocol and a random output
  - Attacker is tasked to distinguish between the two
  - Attacker's advantage should be negligible
  - Security proof of various cryptographic construction rely on this 'game'
- The concept of indistinguishability is important for physical security, too

Security Principles of Information Theory

- Fundamental topics of information theory include coding theory
  - Error Correcting Codes (ECC) and error detection
  - Source coding and compression
  - Post-Quantum Cryptography (PQC) often relies on coding theory
- Information-theoretic security principles
  - Communication link is often modeled as "Wyner's Wiretap Channel"
  - To eavesdrop on channel, Eve is limited by *noise* $\longrightarrow$ *physical* assumption!
- Please note: bits of security are not made all equal:
  - In computational complexity: Shannon entropy bits ('on average')
  - In information-theoretic context: Shannon, or *min*-Entropy, or …
- Relevant for Physical Unclonable Functions (PUFs), Side-Channel Analysis (SCA)
- Comparison of cryptographic and information-theoretic notions here [1]

[1] "Semantic Security for the Wiretap Channel" by Bellare et al. (2012)

Concept of Trust vs. Security

- Trust: a complicated notion related to belief in honesty, reliability, etc. of an entity
- Trusted: a system or component whose failure can break the security policy
- Trustworthy: the degree to which the behavior of the component is demonstrably compliant with its stated functionality
- Demonstrably compliant = having assurance
- A system is defined as *trustable* as long as "*if it always behaves in the expected manner for the intended purpose*" [Trusted Computing Group]

(source: Google)

Motivation for Physical Attacks

- Hack once, own them all, possibly forever
  - Extract firmware or CSPs from one device; own all the other devices
  - Enable subsequent software/firmware analysis
  - Extend functionality, product piracy, to show off, etc.
- Hack what is outside the software scope
  - Low level attack = difficult to counteract in software
  - Increasingly more functionality in hardware, hidden from software
  - Example: actual key storage and encryption in hardware; only API call
- Lack of feedback to manufacturer
  - Device can be attacked without anyone knowing (e.g., unlike websites)
  - No monitoring etc. outside of what shipped within the device
  - Therefore no need to disclose the applied technique!
- Software security 'just' a logical problem
  - Hardware-assisted schemes strong, e.g., pointer authentication, tagged memory
  - Exploits working around these a million dollar business → proof of difficulty

In General: Physical Attack Scenarios and Goals

- Attacker can be legitimate owner of the device
    - Can use device in intended ways
    - Can manipulate, disturb, and observe the device
- Attack may target CSPs, or firmware, etc.
- Revealed information may be used for:
    - Counterfeit product or to modify it in unintended ways
    - Gain unauthorized access to sensitive information
    - Possibly tap into the communication of others
- Physical attack means to use physical properties of the device to break it
- Therefore, typically requires control over the device or close proximity

Quick Facts: Timing Side-Channel Analysis

- The term “Timing Attack” was first introduced at CRYPTO’96 by Paul Kocher (back then: Crypography Research Inc., today: a division of Rambus)
- Few other theoretical approaches without practical experiments; first publicly known practical results in early 1998
- Basic observation: processing time may depend on the data processed
- Absolute duration or differences can be measured
- Practical requirements for attack:
  - Possibility to (indirectly) monitor processed secret data
  - Have a way to automate and record the process
  - Basic computational and statistical tools
  - Either know or suspect properties of the implementation

Example Attack: Electronic Safe Lock

- Suppose you have a safe with an electronic lock with PIN entry
  - In order to open the safe, a secret 8-digit PIN must be entered $(0 \ldots 9)$
  - Each digit is represented by one byte
  - The device has a delayed response upon verification of $100\,\mathrm{ms}$
  - For a brute-force attack: there are 10^8 possible PIN codes
  - Trying all possible PINs (without entering them) already takes 116 days
- Formally secure enough ... but are there better attacks?

(source: lock I recently bought on ebay)

Example Attack: How to Approach

- Suppose that attacker has access to a reference lock
  - Learns the type of 8-bit microcontroller used and its instruction set
  - Is able to automate the PIN entry
- Let us assume the PIN check is done digit by digit (as shown below)
- Early return on wrong digit leads to data-dependent process
- How to attack? Divide-and-conquer strategy!

```
unsigned char verifyPIN(unsigned char Pref[], unsigned char P[]) {
  for(int i=0; i<7; i++) {
    if (P[i] != Pref[i])
      return 0xEE; // return fail code
  }
  return 0x00; // return pass code
}
```

Example Attack: Timing Side-Channel Analysis

- Due to automation, attacker is able to measure time
  - Time between PIN entry to response matters
  - Response can be audible (buzzer), visible (LED), or wire tapped
  - Response detection can be automated
- Attacker selects a fixed PIN and changes only first digit
  - 9x the wrong result (same timing)
  - 1x the correct PIN-digit (different timing)
  - Difference in occurrence gives away important information
  - Likely the correct PIN-digit has a longer response (return on 2nd digit)
- After finding first digit, remaining digits work the same
- Procedure only takes 10x8 = 80 timing measurements!

Example: Possible Countermeasures and Obstacles

- Timing cannot be measured precisely $\rightarrow$ multiple measurements and averaging
- PIN retry counter with additional delay $\rightarrow$ attack slowed down but still works
- PIN retry counter with hard limit of X attempts
  - Consider the business scenario: people forget their PIN and may try too often
  - Fault attacks may help to either skip counter increment or reset to previous state
- You are an uber-programmer and implement an all-digits-at-once comparison
  - Remember, $$$ is important and your boss only approved an 8-bit microcontroller
  - In this case, lack of an all-digit-at-once comparison has little to do with your skills
- Developer of system is constrained by time, money, marketing, physics, …
- Attacker possibly limited by laws of physics only!
- Considering the cost of attacks vs. product difficult

Example Attack: Continued

- Requirement: Runtime of PIN verification must be independent of entered PIN
    - Not easy but possible,e.g., balancing execution time by NOP instructions
    - Other microarchitectural problems: caches, out-of-order execution, etc.
    - Of course, an 8-bit microcontroller has a very simple microarchitecture
- Let’s assume timing behavior has been fixed – problem solved?
- No – most digital circuits are implemented by CMOS logic
    - Power consumption depends on executed instruction and processed data
    - Part of this course to study this type of attack in greater detail
    - For now, we take this for granted (without understanding the ‘why’)

Example Attack: Power Analysis

- Instead of measuring time, attacker measures power consumption
- Attacker selects a fixed PIN an changes only first digit
    - 9x the wrong result (same power signature)
    - 1x the correct PIN-digit (different power signature)
    - Difference in occurrence tells you something
    - Likely the correct PIN-digit has most significant difference
- After finding first digit, remaining digits work the same
- Procedure only takes 10x8 = 80 timing measurements!
- Practical limitations such as measurement noise make this more difficult

Example: Possible Countermeasures and Obstacles

- Solving this dependency of processed data and executed path is difficult
  - Custom logic style needed in hardware (even then, not easy)
  - Masking the actually processed data “somehow”
- Arbitrary timing / random delays make this more difficult, too
- Random processing of digits upon verification (“shuffling”)
- Doing something random requires … randomness!
- True Random Number Generator (TRNG) often not included
- Switched capacitor design (draw power from internal buffer)
- Consider this all solved – are we still able to defeat the lock?

Example: Additional Attacks

- Other ways to deduce similar side-channel information
  - Electromagnetic Emanation (EM)
  - Thermal emission / Temperature
  - Photonic emission
  - Acoustics
- Attacker may not be concerned about learning the PIN
  - Tampering with frequency of circuit (clock glitch) to skip verification
  - Tampering with voltage supply (power glitch) to skip verification
  - For the attacker, only one solution is enough to bypass security
- Why bother with the PIN? A spiker/zapper may initiate bolt movement
- Alternative approach: extract recovery PIN from a reference device

Implementations Matter!

- By way of example, we have seen that
    - Securely implementing a PIN verification is difficult
    - Improper programming was only one of the ways to fail
    - Not knowing processor related dependencies was yet another
    - Perhaps we cared about the wrong part of the system
    - We were not even limited by project deadlines or business aspects
- Non-standard operations: difficult to solve satisfactorily
- Standard operations such as encryption best implemented in hardware
- We have to learn a little bit of crypto to understand the implementation
- NEVER implement security on your own; always have it tested!

Applied Cryptography
Advanced Encryption Standard

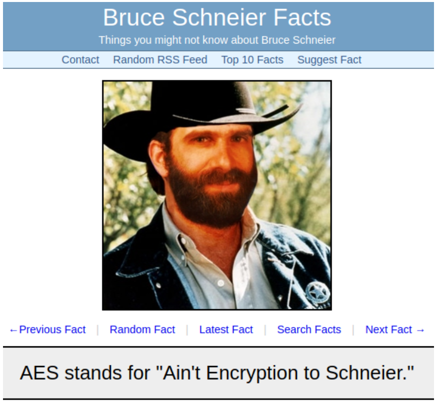
Advanced Encryption Standard (AES)

- Symmetric ‘block’ cipher
- Successor of Data Encryption Standard (DES)
- Selection through open competition
- Selection by US National Institute of Standards and Technology (NIST)
- Likely to be the dominant cipher for the next two decades
- Much more secure and efficient than DES or 3DES
- cf. NIST FIPS 192 “Advanced Encryption Standard” [2]

[2] `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf`

AES Timeline

- AES competition announced in January, 1997
- 15 candidates submitted by August, 1998
- 5 finalists announced in August 1999:
  - Mars, IBM Corporation
  - RC6, RSA Laboratories
  - Rijndael, Joan Daemen and Vincent Rijmen
  - Serpent, Eli Bihman et al.
  - Twofish, B. Schneier et al.
- Rijndael standardized as AES on October, 2001

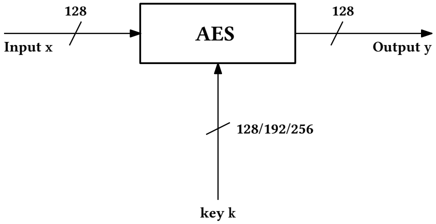


AES stands for "Ain't Encryption to Schneier."

(source: schneierfacts.com)

High-level View on AES

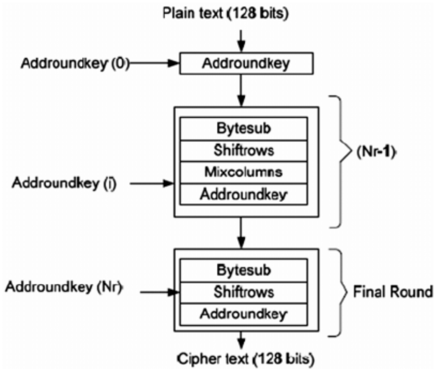
- AES supports 3 key lenghts: 128, 192, 256 bit
- 128 bit key already resistant against brute-force attack for foreseeable future
- No relevant weaknesses despite 20+ years of research in the crypto community
- NSA allows AES for classified documents up to TOP SECRET; similar to other countries



https://en.wikipedia.org/wiki/Commercial_National_Security_Algorithm_Suite

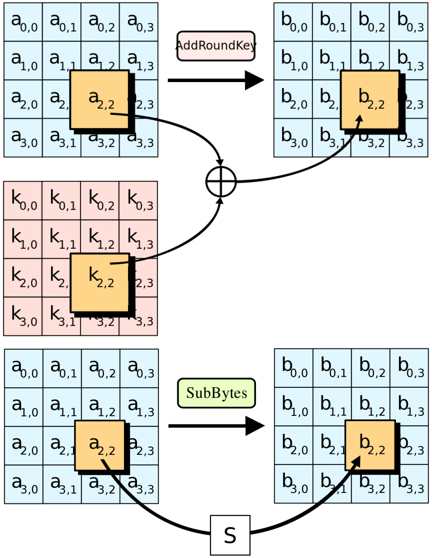
AES Basic Structure

- AES comprised of a Substitution Permutation Network (SPN) repeated for several rounds
- The block size and data path are 128 bit
- Number of rounds: 10, 12, 14 based on selected key length 128, 192, 256
- Notable components:
  - Key Addition: XOR with roundkey
  - Byte substitution: 8x8 S-Boxes
  - Diffusion Layer: create an avalanche effect by spreading influence of each input bit over many output bits
  - Here: diffusion layer created by ShiftRows and MixColumns



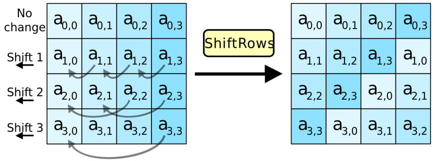
AES AddRoundKey and SubBytes

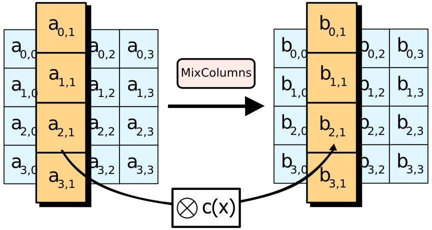
- 4x4 byte-wise matrix to represent internal state
- All fundamental operations with this matrix
- AddRoundKey is a bitwise XOR operation
  - Applies to full width of data path
- SubBytes operates on each byte independently
  - all Sboxes are the same
  - Sbox is inversion in GF(2^8)
  - followed by affine transformation



AES ShiftRows and MixColumns

- ShiftRows
  - Rotates each row by 0, 1, 2, 3 bytes to the left
  - In Software: adjust index
  - In Hardware: just (re-)wiring
- MixColumns
  - Columns represent quarter-state of AES
  - Each column is multiplied with fixed matrix
  - See next slides for details





Closer Look at MixColumns

- MixColumns is a matrix multiplication as follows:

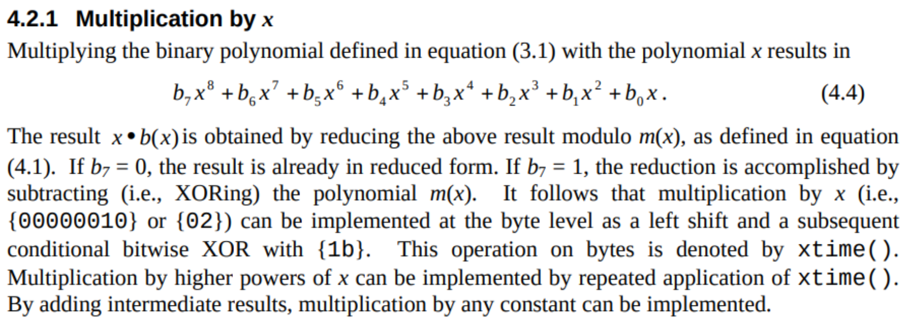
$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

- Resulting in:

$$\begin{aligned} b_0 &= 02a_0 + 03a_1 + 01a_2 + 01a_3 \\ b_1 &= 01a_0 + 02a_1 + 03a_2 + 01a_3 \\ b_2 &= 01a_0 + 01a_1 + 02a_2 + 03a_3 \\ b_3 &= 03a_0 + 01a_1 + 01a_2 + 02a_3 \end{aligned}$$

- Since the elements are in GF(2^8), “special” math applies here
- Plus sign is simply XOR; Multiply by 2 and by 3 needs to be implemented

Multiply by 2 and 3 (from FIPS 197)



4.2.1 Multiplication by *x*

Multiplying the binary polynomial defined in equation (3.1) with the polynomial *x* results in

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x . \qquad (4.4)$$

The result $x \bullet b(x)$ is obtained by reducing the above result modulo $m(x)$, as defined in equation (4.1). If $b_7 = 0$, the result is already in reduced form. If $b_7 = 1$, the reduction is accomplished by subtracting (i.e., XORing) the polynomial $m(x)$. It follows that multiplication by x (i.e., `{00000010}` or `{02}`) can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with `{1b}`. This operation on bytes is denoted by `xtime()`. Multiplication by higher powers of x can be implemented by repeated application of `xtime()`. By adding intermediate results, multiplication by any constant can be implemented.

- To multiply by 2, we need to shift left, check the carry, if the carry is 1, the result must be XORed with 0x1b for polynomial reduction
- To multiply by 3, after multiplying by 2, the result is XORed with the input, i.e., 03a = 02a + a
- Many example AES codes can be found online making use of this (→ homework)

AES Key Schedule

Omitted for reasons of brevity. It is not that important for what we want to do.

AES: How to Implement

- Relatively straightforward for initial functionality
    - Performance may not be well-adapted to targeted platform
    - Very unlikely to obtain a secure implementation
- In Software:
    - Sbox can be implemented as Look Up Tables (LUT)
    - 4 Sboxes and a MixColumns merged into 8-bit to 32-bit LUT called T-table
    - Typically the fastest approach if not constrained by memory (4kb)
- In Hardware:
    - Same as software when not constrained by memory
    - Alternatively, composite field GF multipliers used for Sbox
    - In addition: various implementation styles possible for size/speed
    - Pipelined; iterated round designs: single Sbox, quarter, full

Risks of Look Up Tables . . . Caches

- Most platforms are optimized for performance and low power (not security!)
- Primary two reasons why complex processors use a cache
  - Performance: accessing a data block in memory is significantly slower than a cache access
  - Power consumption: accessing a data block in memory consumes more energy than a cache access
- Caches can be implemented as
  - data cache
  - instruction cache
  - unified data and instruction cache
- Nowadays, computers typically have 3 or more levels of caching!

Timing-Dependency of Caches

- Cache hits and misses affect the timing characteristics of the implementation
- Instruction flow may be constant, but data-fetch process is not
- If the crypto algorithm is implemented with LUTs and caches exist
  - Knowing if a cache miss or hit gives away information
  - Prevention is difficult, e.g., loading LUT in cache completely
- Information leak is essential to this type of attack
- Similar to pointer leaks for software exploits, this improper caching behavior can be provoked with suitable gadgets that interfere with regular caching

Types of Cache-based Attacks

- Attacks are classified by how information leak is observed
- Trace-driven attacks
  - Trace can be recorded including hits/misses: Miss Hit Miss Miss Hit Hit …
  - Trace can be a spy process or, e.g., measured power/emanation
- Time-driven attacks
  - Execution time is different for each run executed
  - Execution time is measured …AND precise enough
  - Enables remote attacks over the network
- Access-driven attacks
  - The attacker tampers with the cache directly
  - By deliberately injecting suspected information or garbage into the cache, the follow-up behavior of the targeted process can be observed directly
- AES-specific attacks based on partially loaded LUTs called: “Cache-Collision”

Example: How to Run Time-Driven Attack

- Adversary model:
  - Attacker only records overall time for encryption
  - Attacker controls plaintext input
- Attack strategy:
  - Key is fixed and target of the attack
  - Collect timings $T = \{t_1, t_2, \ldots, t_n\}$ for n encryptions of plaintext $P = \{p_1, p_2, \ldots, p_n\}$
  - Each key byte can be attacked separately following a divide-and-conquer strategy
  - Let us consider one key byte

How to Run the Attack

- Attacker guesses a value K as key candidate for the key
    - Create two groups: group1 and group0
    - for $i = 1$ to n compute Sbox(firstbyte(p_i) XOR K) – this will be our 'hypothesis'
    - If MSB of result is 1, the corresponding timing t_i goes to group1, otherwise to group0
    - On average, the timings of group1 should be higher than of group0
    - Averaging helps to get this information and reduce noise
    - The difference between averages of two groups is assigned guessed key byte K
- This step needs to be repeated for all key candidates $K = 0 \dots 255$
    - The highest difference between the averages shows the most probable value candidate for the key

$\longrightarrow$ You will perform this as homework assignment

$\longrightarrow$ Closely following this 'recipe' will lead to success

Summary of AES and Timing-Dependency

- Timing-only attacks on symmetric cryptography primitives a niche topic
  - Complex devices: AES-NI or similar instruction sets offer constant runtime
  - Embedded devices: only limited number of scenarios where resistance against timing attacks is required but no resistance against power analysis is necessary
  - Aspects of timing are an integral part of power analysis
- Timing information from erroneous behavior combined with error messages can still have devastating effect, even when cryptographic primitive itself is secure!
  - "Padding Oracle" attacks, e.g., AES in Cipher-Block-Chaining (CBC) mode
  - Bleichenbacher Attack (1998), Lucky Thirteen Attack (2013), POODLE (2014)
  - Follow-up attack in 2016 enabled by initial patch in OpenSSL to fix Lucky Thirteen
- In addition: timing-dependency due to improper performance optimization has been the foundation for the famous SPECTRE microarchitectural attack (2018) which is a timing side-channel attack many thought not being practically possible

Summary of AES and Timing-Dependency

- Countermeasures:
  - Security-aware programming; including proper error-handling
  - No branch operations depending on sensitive data either directly or indirectly
  - Avoid instructions with data-dependent timing when processing secret data
  - Be aware of compiler optimizations and external libraries
- Make sure LUTs are completely cached (or not cached at all)
- Unknown cache architecture makes this more difficult (set-associative caches)
- Alternatively: avoid the LUT and compute result (better)
- Make use of dedicated hardware accelerators if available
- Limit precision of information available to attacker and access
- Generally avoid multi-tenant environments for highest security

Thank you for your attention!
Questions?