ECE 455 – Module 5

Security

Spring 2023

Murray Dunne – mdunne@uwaterloo.ca

Slides Acknowledgement

Some material in these slides is based on slides from:

- Prof. Sebastian Fischmeister
- Prof. Carlos Moreno

Other material used with citations

Disaster of the Day – “Kia Challenge” (2022)

Embedded Systems Security

- How is this different from general purpose computer security?
  - Server security? IT security? Web security?

- The attacker is *physically present* with our device

- Mostly C/C++/Assembly code
  - Manual memory management
  - Stack management
  - Simple memory model

Buffer Overflow

- Common attack on C/C++/Assembly programs
- Example: wireless door lock

```
void read_data_from_fob (char * wrless_data)
{
    char buffer[33];  // hex-encoded 256-bit block + \0

    strcpy (buffer, wrless_data);
    if (decrypt (buffer, SECRET_KEY) == DECRYPT_OK)
    {
        unlock_door();
    }
}

void unlock_door()
{
    output_port (DOOR_PORT_CMD, COMMAND_BITPATTERN);
}
```

Solution Attempt

- Transmit the bit pattern required to unlock from the fob

```
void read_data_from_fob (char * wrless_data)
{
    char buffer[33];  // hex-encoded 256-bit block + \0

    strcpy (buffer, wrless_data);
    if (decrypt (buffer, SECRET_KEY) == DECRYPT_OK)
    {
        unlock_door(extract_code(buffer, SECRET_KEY));
    }
}

void unlock_door(uint8_t bit_pattern)
{
    if(bit_pattern == 0x5A)
    {
        output_port (DOOR_PORT_CMD, COMMAND_BITPATTERN);
    }
}
```

Solution Attempt 2

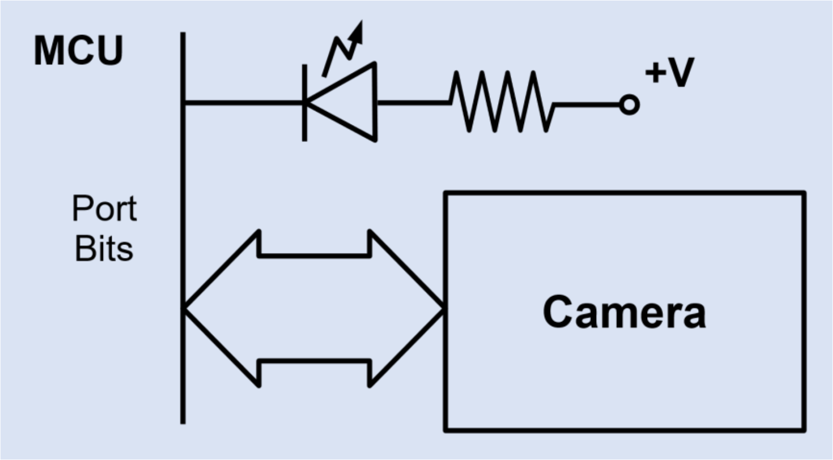
- Transmit the bit pattern required to unlock from the fob
  - The hardware reads it

```
void read_data_from_fob (char * wrless_data)
{
    char buffer[33];  // hex-encoded 256-bit block + \0

    strcpy (buffer, wrless_data);
    if (decrypt (buffer, SECRET_KEY) == DECRYPT_OK)
    {
        unlock_door(extract_code(buffer, SECRET_KEY));
    }
}

void unlock_door(uint8_t bit_pattern)
{
    output_port (DOOR_PORT_CMD, bit_pattern);
}
```

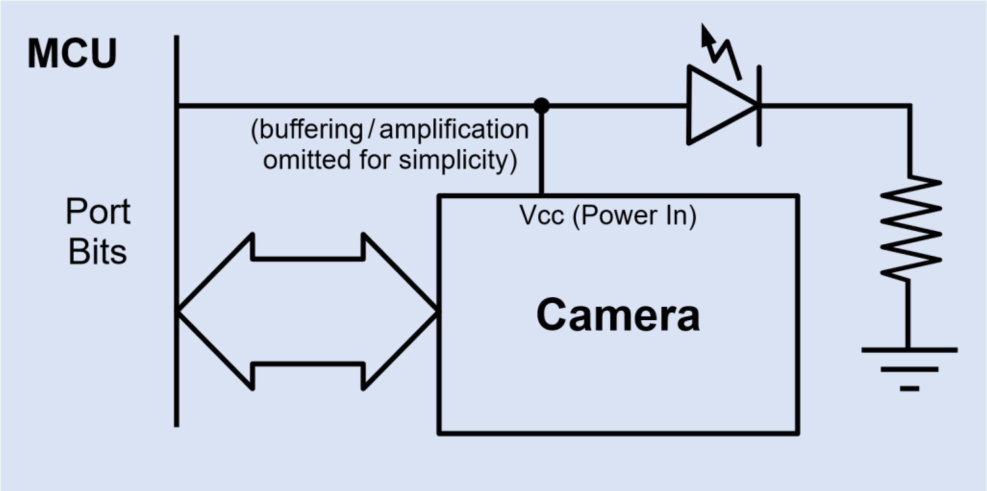
Hardware Solutions are Better

- If your hardware can provide security, do it
- Example: webcam LED

Independent LED:



Better idea:



Apple *Almost* Got this Right

- But not quite…
  - Camera microcontroller connected to image sensor
  - LED on the power-mode pin of the image sensor
    - If the image sensor was not in low-power mode then the LED was on
  - Problem:
    - There was a “ignore low-power mode” setting on the image sensor
      - This disabled the power-mode pin and therefore kept the LED off
    - Malicious firmware on the microcontroller could set this register
      - Could do this without root permissions on OS X
      - At *any* time

- This was patched a long time ago

Disaster of the Day – Patriot Missile Defense System (1991)

Cryptography – Whirlwind Tour

- This is a *very* high-level view!

- **Confidentiality**
  - Can some else other than the intended recipient read the data?
- **Authenticity**
  - Is the person sending/receiving the data who they say they are?
- **Integrity**
  - Is the data I’m receiving the data that was actually sent to me?

Types of Cryptography

- **Symmetric key** cryptography
  - Both parties share a secret
- **Asymmetric key** cryptography
  - Also called **public key** cryptography
  - Parties do not share the same secret

Disaster of the Day – Soviet Gas Pipeline Explosion (1982)

Symmetric Key

- Sender and recipient have pre-shared some secret key K
- Provides **confidentiality**
- Example (bad):
  - To encrypt some secret value X calculate $\mathrm{C} = X + K$
  - We say:
    - X is the **plaintext**
    - C is the **cyphertext**
    - K is the **key**
  - Addition is really bad for several reasons
    - Once you guess one letter you know K and therefore all letters
    - The pattern of input is preserved
      - Very easy to guess from pattern

Symmetric Key

- Example (better):
  - To encrypt some secret value X_i calculate $\mathrm{C_i} = X_i \oplus K_i$ ($\oplus$ is xor)
    - Then move on to the next key K_{i+1} for the next number X_{i+1}
  - This is very effective (theoretically unbreakable)
  - But requires K just as long as X
    - Infeasible for all but the simplest cases
  - This is called a **one-time pad**

- In practice:
  - Use a modern symmetric key algorithm like **AES** (**Advanced Encryption Standard**)
    - Complex intermixing of bits in chunks
    - Comes built into many modern microprocessors in hardware (including ARMv8)

Course Evaluations

`evaluate.uwaterloo.ca`

Problem with Symmetric Key Cryptography

- You have to share a secret
  - Somehow prearrange the transfer
    - This can be difficult
  - Does not work for entities that have never had any prior contact

  - This isn't a huge barrier in embedded systems
    - We built the system in our facility so we can give it a key at manufacturing time
    - Better keep that key *very* safe...
      - Perhaps in a **Hardware Security Module** (**HSM**)
        - Physical module to protect data from reading, tampering, or other interference

Public Key (Asymmetric Key)

- Use two different keys instead!
  - One key to encrypt, one to decrypt
- The two keys are related
  - What is encrypted by the encryption key can only be decrypted by the corresponding decryption key
  - The relation is non-obvious
    - It must be computationally infeasible to determine the decryption key from the encryption key
      - The other way around is fine though
  - The encryption key is made public
    - It is the **public key**
  - The decryption key remains a secret
    - It is the **private key**

Public Key Continued

- Everyone shares their public keys around
  - I know if I encrypt something with your public key that you are the only person that can decrypt that message
    - Provides **confidentiality** in one direction
  - So long as I know the key I have for you is actually *your* public key
    - We need a bit more to protect from **man-in-the-middle** attacks

Public Key Continued 2

- Two main categories
  - **Diffie-Hellman** based
    - The Diffie-Hellman algorithm itself is actually a secret exchange algorithm, however it formed the basis for many public key cryptosystems
    - Relies on the difficulty of the discrete logarithm problem
  - Rivest–Shamir–Adleman (**RSA**) based
    - Relies on the difficulty of factoring extremely large integers

Digital Signatures

- Sign some string with your private key
  - Anyone with your public key can verify that only you could have signed that document
    - So long as you keep your private key private, nobody else can produce a signature that matches your public key
  - Provides **authenticity**
    - Only someone with the corresponding private key could have sent me this data
  - Provides **integrity**
    - The signature is only valid for that string
      - If the string is tampered with, the signature won’t match

Public Key Infrastructure

- The string we sign could be *someone else's public key*
  - By signing their key I am vouching for them
  - We can chain keys together this way to get a web of trust
  - Give signatures a time limit, and revoke trust as necessary
    - Go back up the chain and make sure all the vouches are still valid

The Discrete Logarithm Problem

- Given
  - base g
  - modulus m
  - y where $y = g^x \bmod m$
- Determine x
- Example: for $93 = 17^x$ mod 100 solve for x
  - Try it!

- We need to use very large numbers for this to work
  - Hundreds or thousands of bits
  - Except g which is commonly 2 or 3 or 5 for DH key exchange
    - Might be large for other algorithms

The Discrete Logarithm Problem Continued

- How do we calculate $g^x \bmod m$ if we do know x
  - We're the intended recipient, so we know x
- Use the **square and multiply** algorithm

```
r = 1
for (i = num_exponent_bits - 1; i >= 0; i--)
{
        r = r^2 mod m
        if((x >> i) & 1)
        {
                r = (r * x) mod m
        }
}
```

Disaster of the Day – Mir EP-3 Soyuz TM-5 (1988)

Side-Channel Analysis

- What if we could bypass the “mathematical” security entirely
  - Instead, we observe the byproducts (side-effects) of the computation
    - This gives us extra information about what the system is doing
  - The hope is these side-effects exhibit a correlation with the secret data
    - These could be exploited to break the system without breaking the math
  - These attacks work well when the attacks has physical access to the device performing the cryptographic computations
    - Which is often the case in embedded systems

Side-Channel Analysis Continued

- Examples of side channels:
  - Timing
  - Power consumption
  - Electromagnetic emanation
  - Sound (yes, acoustic vibrations from a chip)
  - Heat/Infrared imaging
  - Cache

Timing Attacks

- Consider the following (bad) password validation

```
bool is_password_ok(char* username, char* pwd) {
        char* actual_pwd = get_pwd_from_db(username);
        return strcmp(actual_pwd, pwd) == 0;
}
```

- What is its *exact* execution time?
    - It depends on the user’s password!

Timing Attacks Continued

- How do we exploit this?
  - We need unlimited attempts at login

- Start with a 1-character password
  - Send all possibilities
    - The correct character means the string compare will take slightly longer
    - Let’s say its “a”
  - Now repeat for “aa”, “ab”, “ac”, etc. for all 2-character passwords
    - The correct second character means the string compare will be even longer
  - Continue until you have the whole password

Timing Attack Countermeasures

- What if we check the length matches first to counter this?
    - Doesn’t work! Just send a password of each length until it runs longer
        - If it ran longer, it means you passed the check and you have the right length
- Instead, only do fixed length computations with sensitive data
    - Hash passwords first, compare fixed length

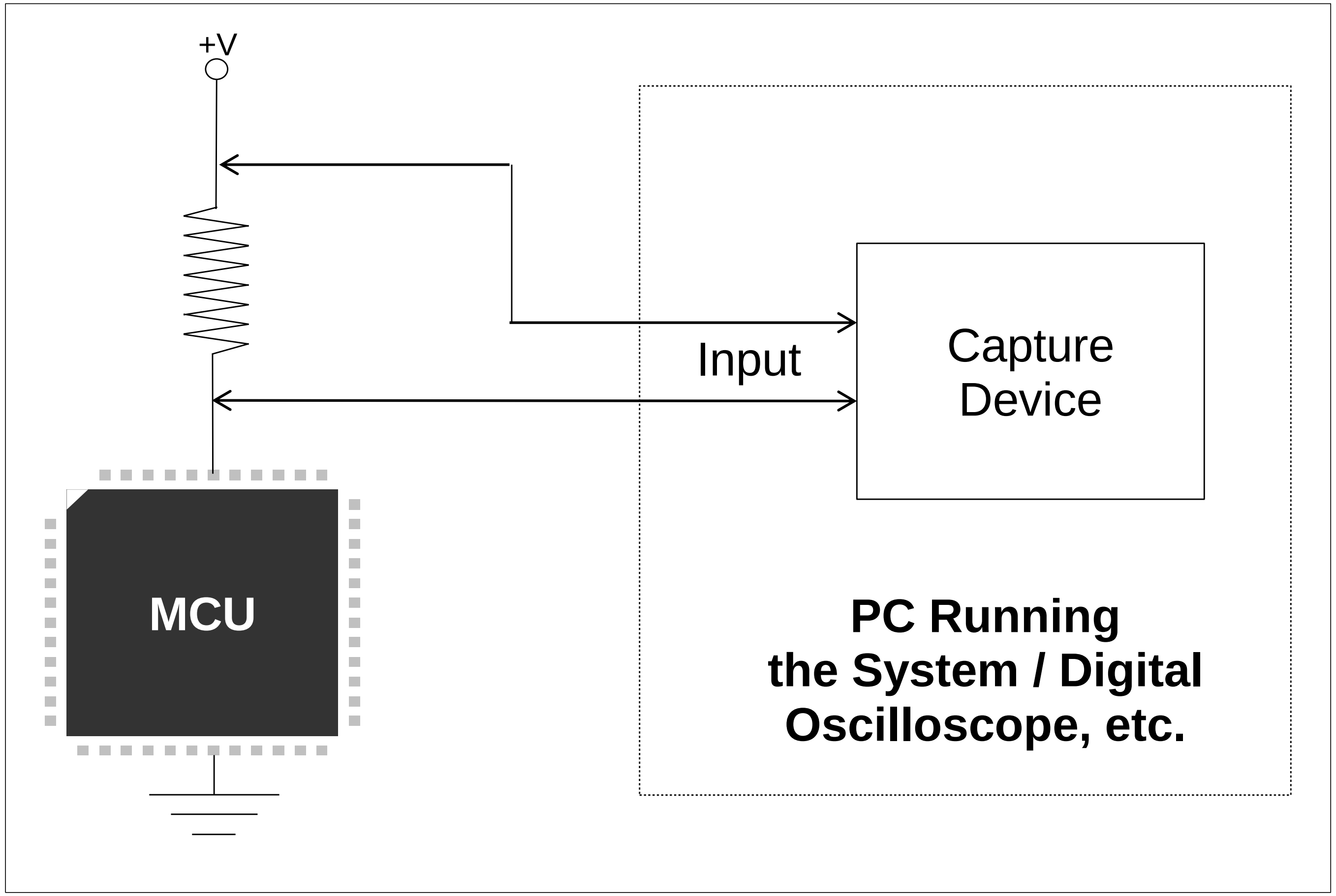
Power Side-Channel Analysis

- Exploit the relationship between the operations the CPU is performing and the data it is operating on

- Two categories:
  - **Simple Power Analysis** (**SPA**)
  - **Differential Power Analysis** (**DPA**)

Power Side-Channel Analysis - Setup

- How do we obtain a power trace?



Simple Power Analysis

- The power consumption of the CPU directly reveals information
  - Perhaps the power trace is:
    - Higher amplitude for a 1 bit than a 0 bit
    - Longer periods of high voltage for a 1 bit than a 0 bit
- Kocher et. Al *Differential Power Analysis (1999)*

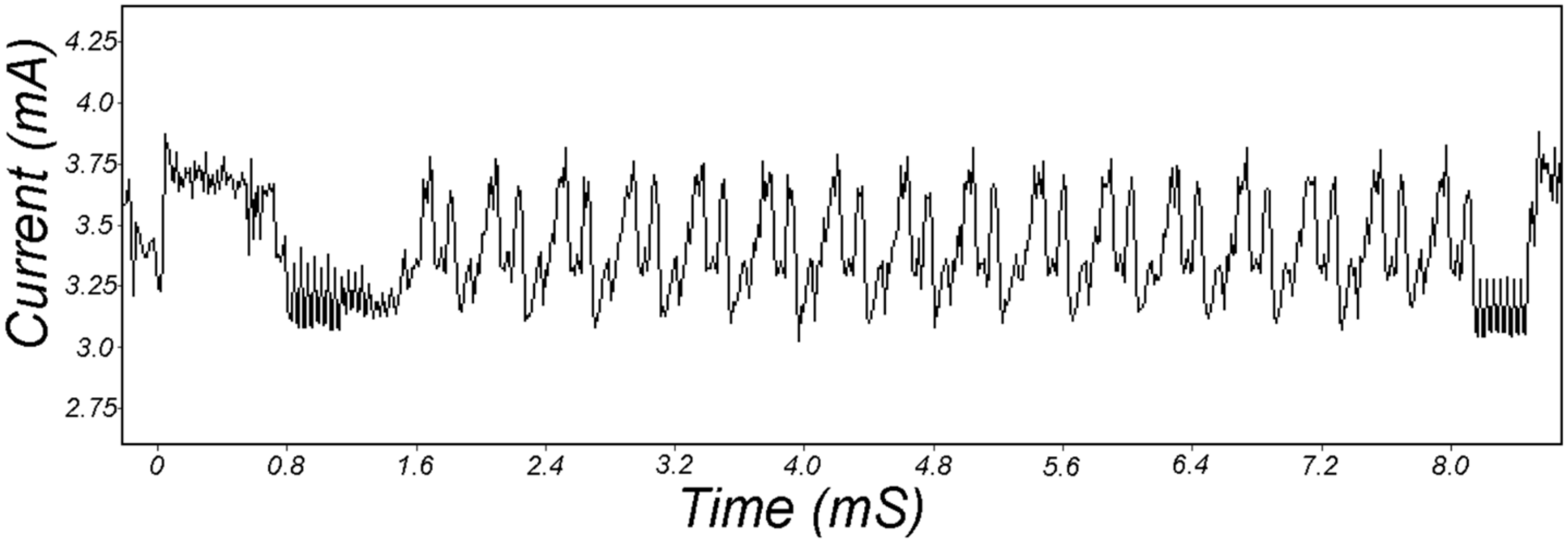
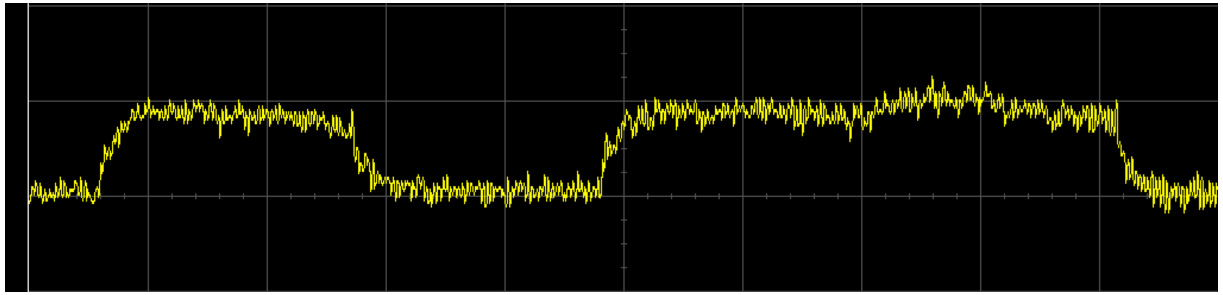


Figure 1: SPA trace showing an entire DES operation.

Simple Power Analysis Example

- Think back to square and multiply
- Power trace will reveal the exponent in a very obvious way
  - When the exponent bit is 0, we see a squaring operation
  - When the exponent bit is 1, we see a squaring followed by a multiplication

```
r = 1
for (i = num_exponent_bits - 1; i >= 0; i--)
{
        r = r^2 mod m
        if((x >> i) & 1)
        {
                r = (r * x) mod m
        }
}
```



http://en.wikipedia.org/wiki/Side_channel_attack

Simple Power Analysis Countermeasures

- Don’t have data conditional on execution!
  - Do the same computation in all cases, just with different inputs
  - Big performance penalty
- For square and multiply, always do the multiplication

```
r = 1
for (i = num_exponent_bits - 1; i >= 0; i--)
{
        temp[0] = r^2 mod m
        temp[1] = (temp[0] * x) mod m
        r = temp[(x >> i) & 1)]
}
```

Differential Power Analysis

- Say we have a device that implements square and always multiply
  - Can we still break the system through power analysis?

- Idea: exploit tiny differences correlated to the actual bits of the data
  - Recursive definition:
    - Assume the attacker has guessed the first (least significant) G bits of x
    - Now they want to calculate bit $G + 1$
    - The attacker can determine all the intermediate results up to the G^{th} iteration of square and always multiply
  - This means if we execute up to iteration G multiple times with the same first G bits all the power traces will be fully correlated up to iteration G then uncorrelated after

Differential Power Analysis Continued

- Idea: what if we execute multiple exponentiations for different data g but with the same exponent x
  - Group these traces where the result y has some specific bit value
    - Perhaps group by bit n
    - You'd have a set of traces that come from a computation that results in $y_n = 0$
    - And a second, disjoint set that come from computations that resulted in $y_n = 0$
  - To do this you need
    - Ability to send input data g to the system
    - Access to output y
    - Knowledge of m
      - This is normally public
  - Remember, we're trying to find x (the key/exponent)

Differential Power Analysis Continued 2

- To determine bit $G + 1$ we guess some value for that bit
- Based on that guess, group the power traces
  - Group based on the intermediate result we know up to bit $G + 1$
  - Multiple power traces are recorded with random inputs
    - All grouped together based on this predicted bit value

Differential Power Analysis Continued 3

- If the guess was correct, the groups will be internally correlated
  - The groups will be a “good classification” based on the result

- If the guess was incorrect, then the groups will be completely random
  - There’s no relation between the value of y and that split of the power traces
    - Because the guessed x was wrong

- If the guess was correct, then that’s the bit
  - If the guess was incorrect, then the other bit value was correct
- Repeat recursively