Cryptographic Side-Channels on Embedded Devices

Tjerand Silde, PONE Biometrics

PONE
BIOMETRICS

Introduction

Associate Professor in Cryptology at NTNU

Working on quantum-safe cryptography and privacy

Security and Cryptography Expert at Pone Biometrics

Teaching a course on “Secure Cryptographic Implementations”

Working on FIDO, secure authentication, biometrics

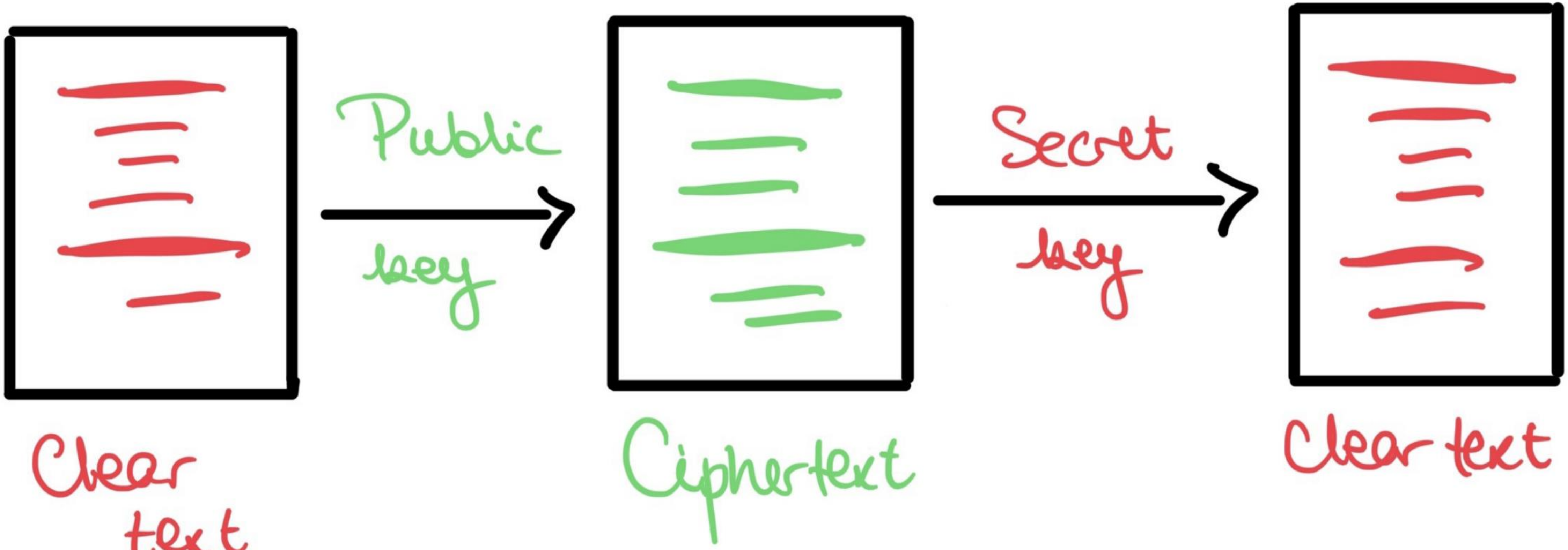
Supervising master’s and PhD students in cryptography

Outline

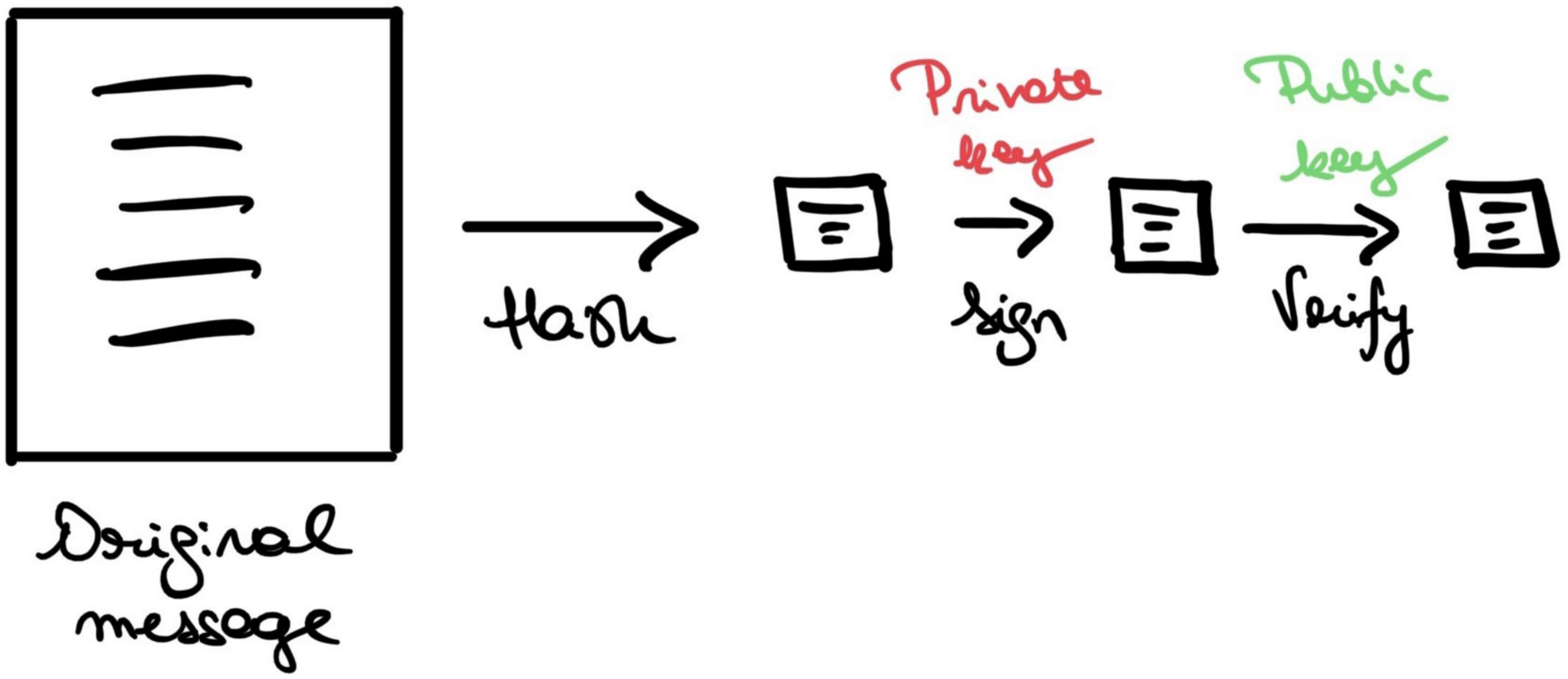
- Cryptography today
- Black Box vs Leakage
- Side-Channel Attacks
- Mitigation Techniques
- Quantum-Safe Crypto

Cryptography Today

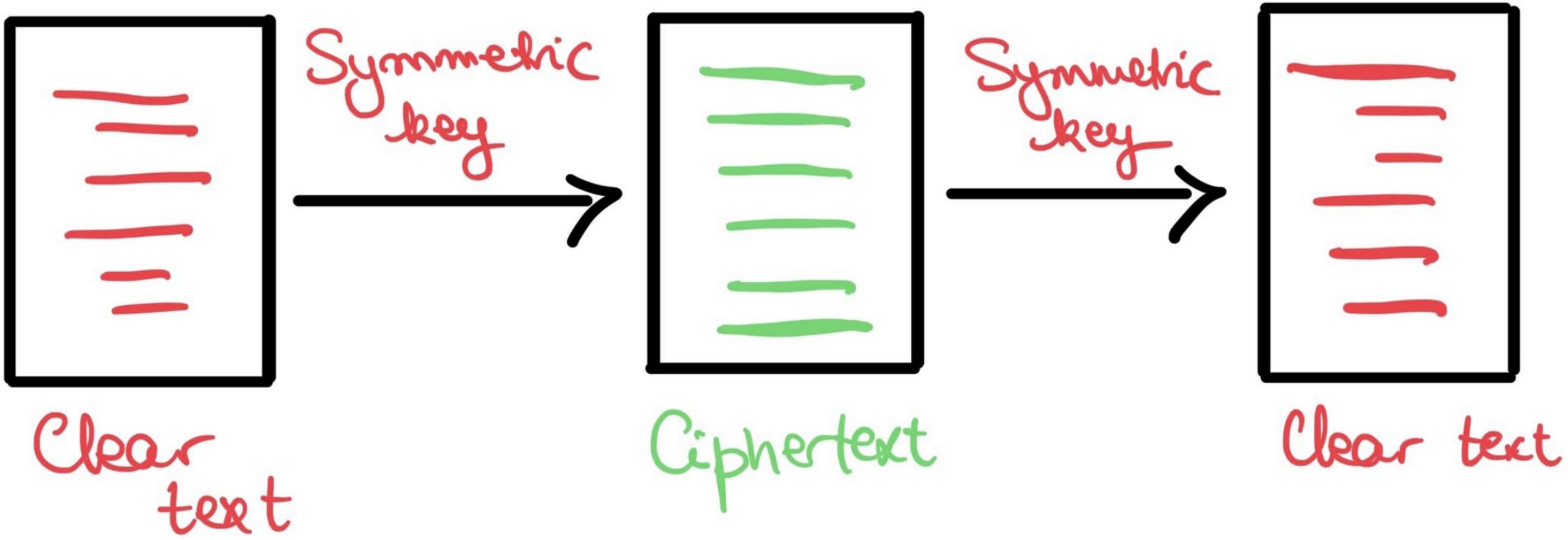
Cryptography Today – Public Key Enc



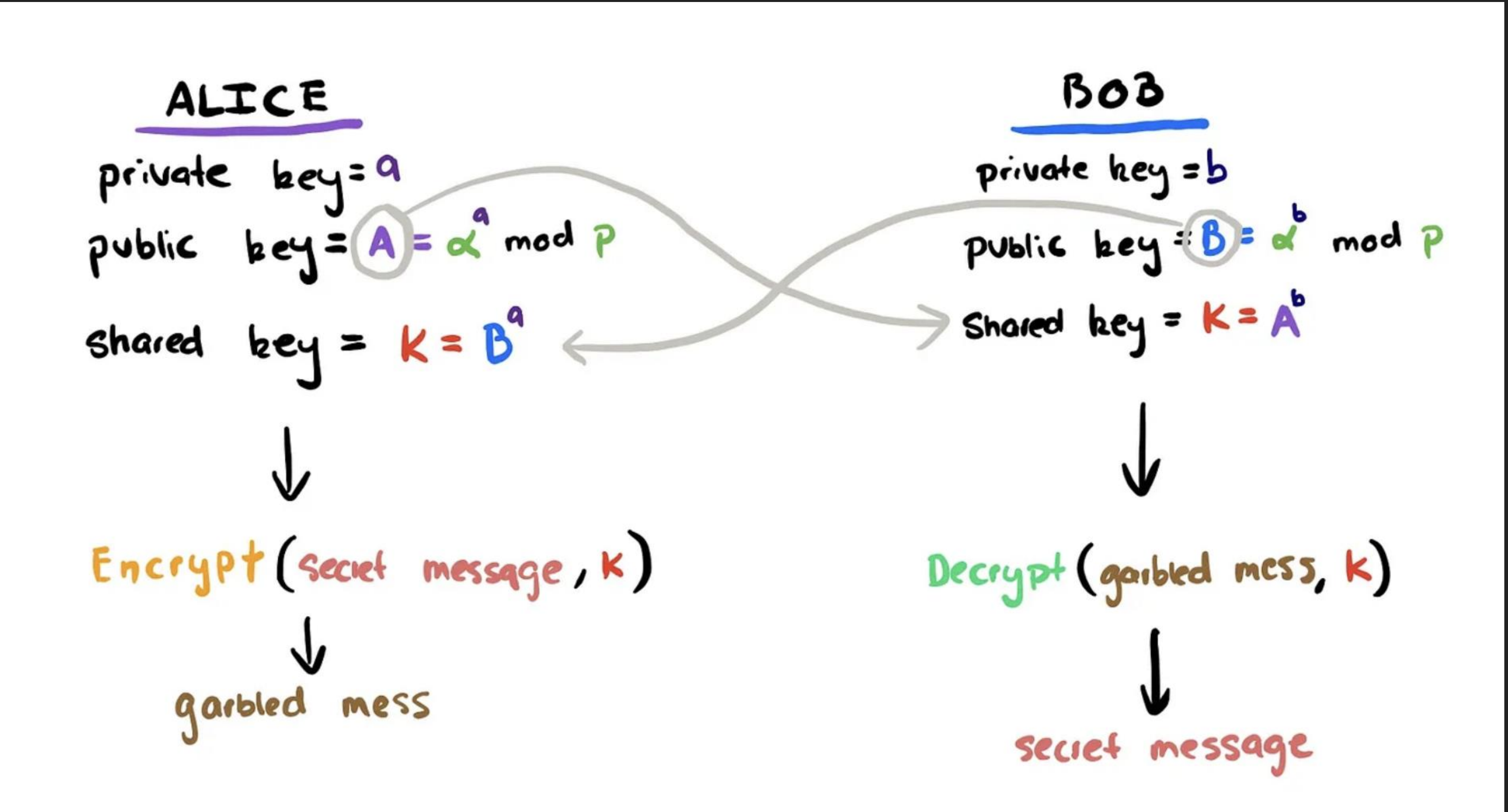
Cryptography Today – Digital Signatures



Cryptography Today – Symmetric Key



Cryptography Today – DH + AES



Black Box vs Leakage

Black Box vs Leakage

We design cryptographic schemes
to follow Kerckhoff's principle:

If everything about the scheme,
except for the key, is known, then
the scheme should be secure.

Black Box vs Leakage

We study black-box algorithms that take some input and give some output, proving that they are secure with respect to the algorithm description and the public data.

Black Box vs Leakage

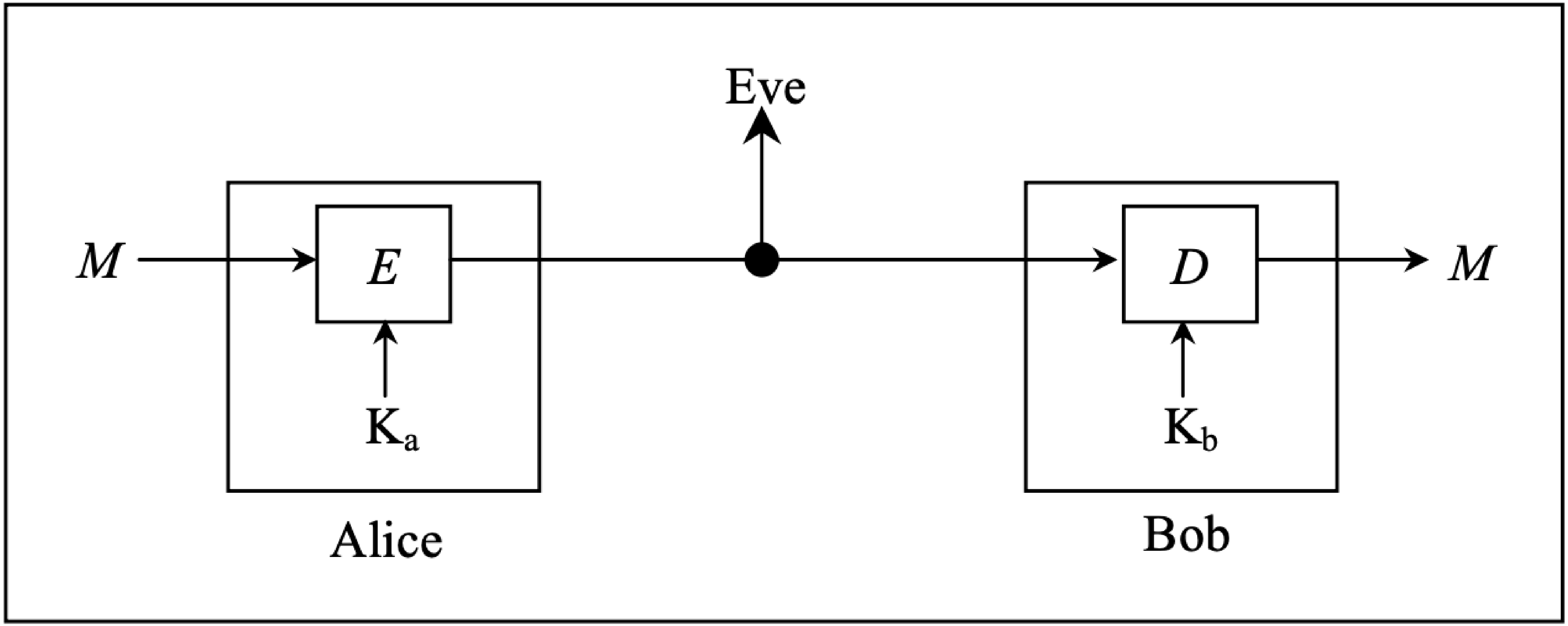


Figure 1: The traditional cryptographic model

Black Box vs Leakage

In practice, it matters how these algorithms are implemented and what kind of information the physical system leaks about the inner workings of the algorithm computing on secret data.

Black Box vs Leakage

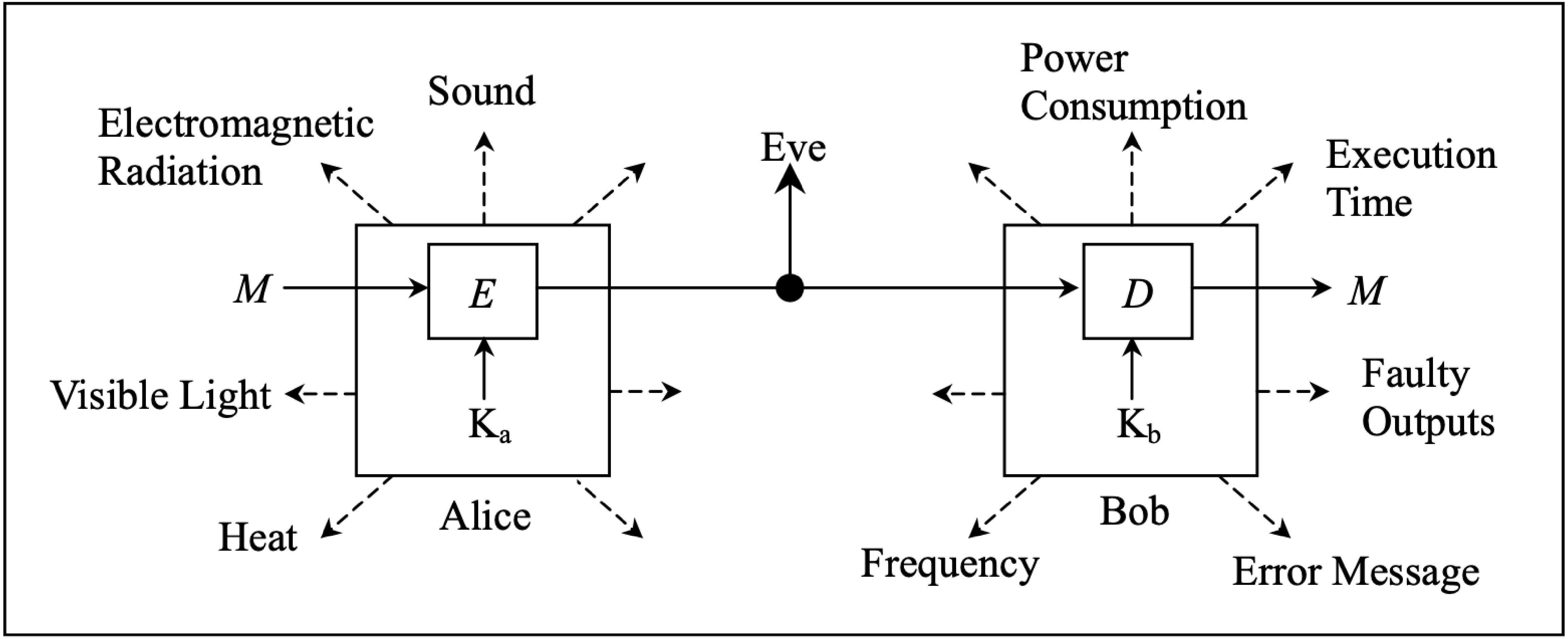


Figure 2: The cryptographic model including side-channel

Black Box vs Leakage

This leakage is more accessible on embedded devices, since they often expose the CPU, run few processes simultaneously, and sometimes even get their resources from the environment.

Password Example

```
def isCorrectPassword(pw, user_input):
    if len(pw) != len(user_input):
        return False

    for i in range(len(pw)):
        if pw[i] != user_input[i]:
            return False

    return True
```

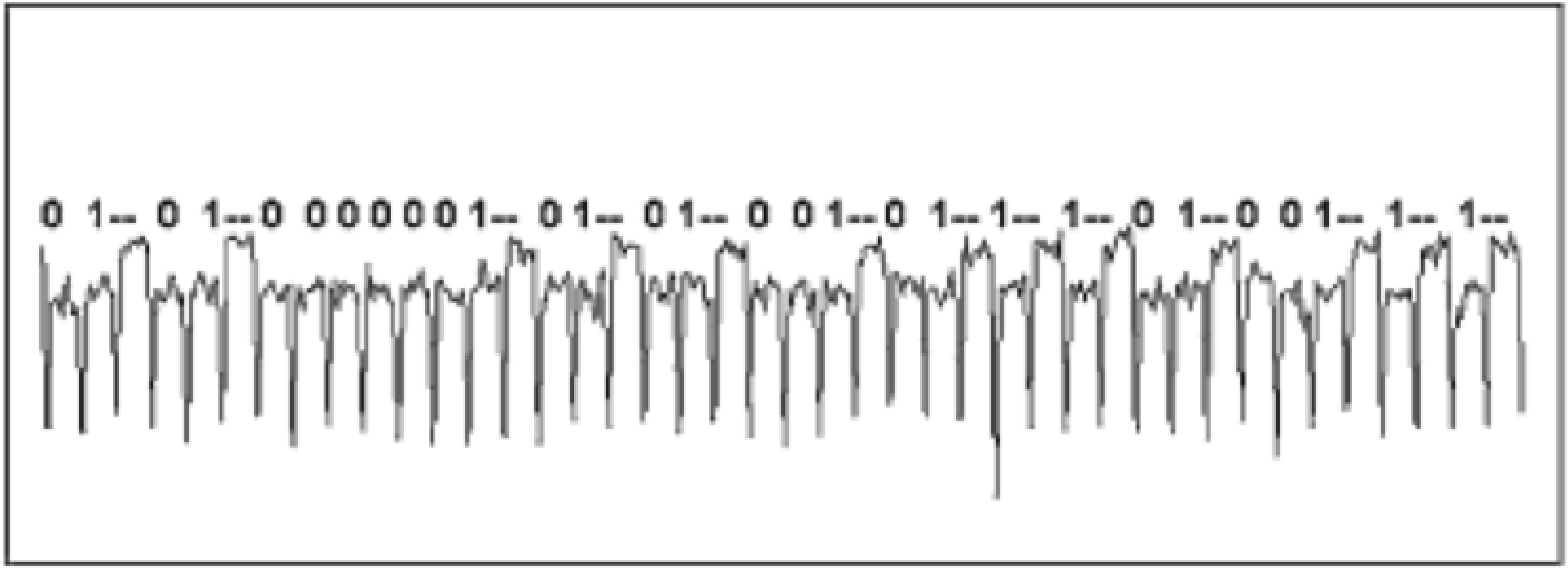
Side-Channel Attacks

Side-Channel Attacks

Side-channel attacks exploit physical leakage in an implemented scheme to extract secret keys.

We use statistical methods to analyze the leakage of one or more traces to break it.

Side-Channel Attacks



Side-Channel Attacks

We usually classify them as:

- Remote vs physical attacks
- Passive vs active attacks
- Invasive vs non-invasive attacks

Side-Channel Attacks - AES

```
def encrypt(key, plaintext):

    # AddRoundKey for initial round
    ciphertext = AddRoundKey(plaintext, key[0])

    for i in range(1, rounds):
        ciphertext = SubBytes(ciphertext)
        ciphertext = ShiftRows(ciphertext)
        ciphertext = MixColumns(ciphertext)
        ciphertext = AddRoundKey(ciphertext, key[i])

    # Final round (no MixColumns)
    ciphertext = SubBytes(ciphertext)
    ciphertext = ShiftRows(ciphertext)
    ciphertext = AddRoundKey(ciphertext, key[rounds])

    return ciphertext
```

Side-Channel Attacks - AES

From a power trace, we can easily:

- See how many rounds are computed
- See which operation is computed
- Compare known traces with the first round

Side-Channel Attacks - AES

Potential leakage:

- AddRoundKey computation might leak HW
- SubBytes is a non-linear operation (inverse)
- MixColumns is a polynomial multiplication
- Algebraic operations are computed over a field

Side-Channel Attacks - PKC

We need to compute modular exponentiations in the DH and RSA public key cryptosystems.

Since the exponent is the secret key, we must protect it against side-channel attacks.

Side-Channel Attacks - PKC

```
    # compute m = c**d mod n
    def squareAndMultiply(c, d, n):
        m = c

        for i in range(len(d)):
            m = m * m % n

            if ( d[i] == 1 ):
                m = m * c % n

        return m
```

Side-Channel Attacks - PKC

```
# compute m = c**d mod n
def squareAndAlwaysMultiply(c, d, n):
    m, x = c, c

    for i in range(len(d)):
        m = m * m % n

        if ( d[i] == 1 ):
            m = m * c % n

        else:
            x = m * c % n

    return m
```

Side-Channel Attacks - PKC

However, dummy operations might leak memory information, consume different amounts of resources, be skipped by “smart” compilers, or be exposed by fault injections.

Side-Channel Attacks - PKC

```
# compute m = c**d mod n
def MontgomeryLadder(c, d, n):
    m1, m2 = c, c * c % n

    for i in range(len(d)):

        if ( d[i] == 1 ):
            m1 = m1 * m2 % n
            m2 = m2 * m2 % n

        else:
            m2 = m1 * m2 % n
            m1 = m1 * m1 % n

    return m1
```

Side-Channel Attacks - PKC

This makes the algorithm regular and the output dependent on every operation.

However, given many traces depending on the same key applied to adaptively chosen inputs, it might still leak information about the key.

Mitigation Techniques

Mitigation Techniques

Some standard approaches:

- Constant time (secret independent) code
- Randomization of (secret) computation
- Splitting the secret into several parts
- Checking that outputs are correct/valid
- Add noise to the computation (delays)

Mitigation Techniques - AES

We use d-order masking by splitting the key:

- Linear operations are easy, non-linear not
- AddKey, ShiftRows, MixColumns are linear
- SubBytes is not linear: requires extra work
- Statistical analysis is exponential in d
- Added work scales with d log d operations

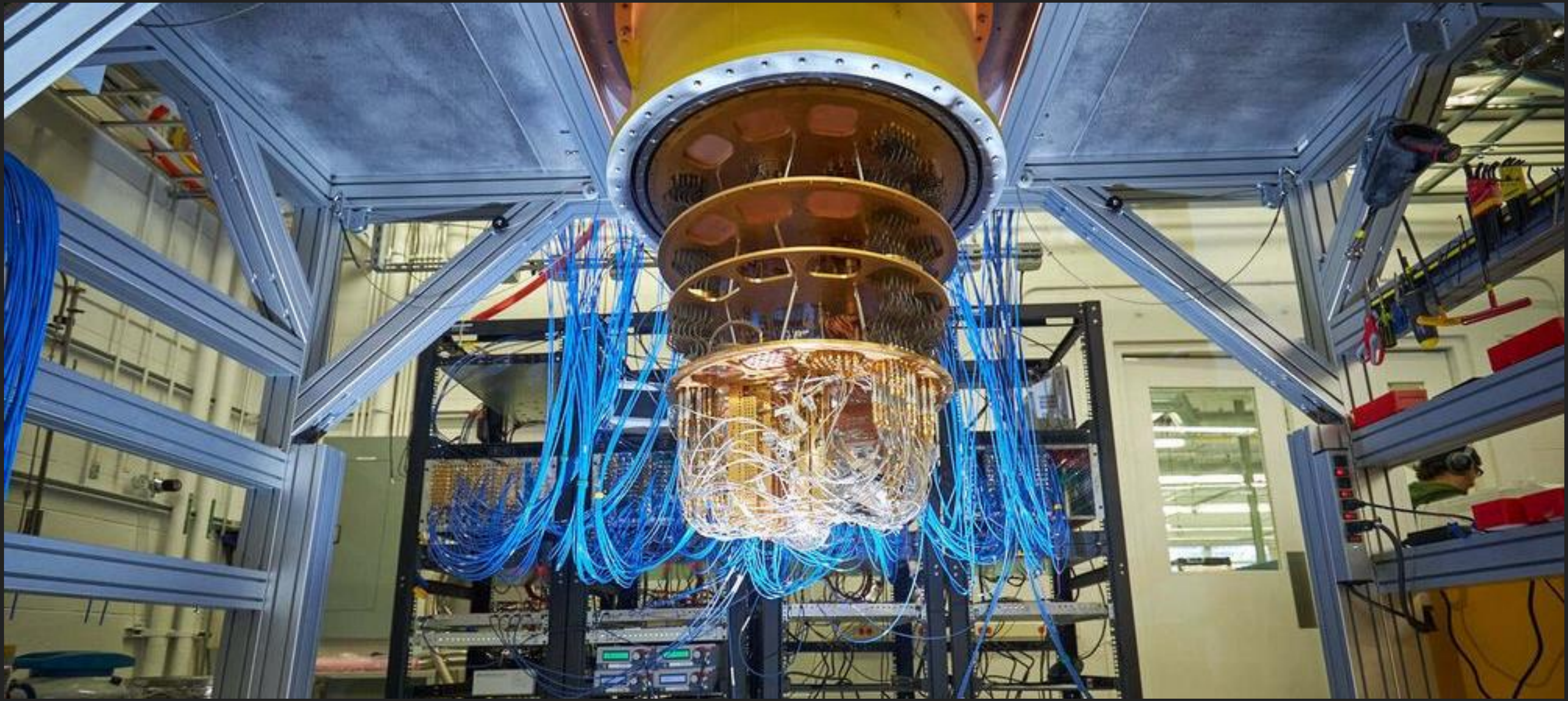
Mitigation Techniques - PKC

We use algebraic countermeasures:

- Ensure modular operations are constant
- Use regular algorithms like Montgomery
- Randomize base elements and exponents
- Verify that inputs and outputs are valid
- (ECC arithmetic might depend on curve)

Quantum-Safe Crypto

Tomorrow: Quantum Computers



Cryptographic Algorithms

RSA Encryption and Signatures,
(EC) Diffie-Hellman Key Exchange,
(EC) Digital Signature Algorithm,
(EC) ElGamal Encryption, Pairings.

Symmetric encryption like AES,
Hash functions like SHA2/3,
MAC schemes like HMAC.

Quantum Algorithms

Shor's Algorithm can be used to efficiently find the periodicity of a function and can be applied to factoring and computing discrete logarithms.

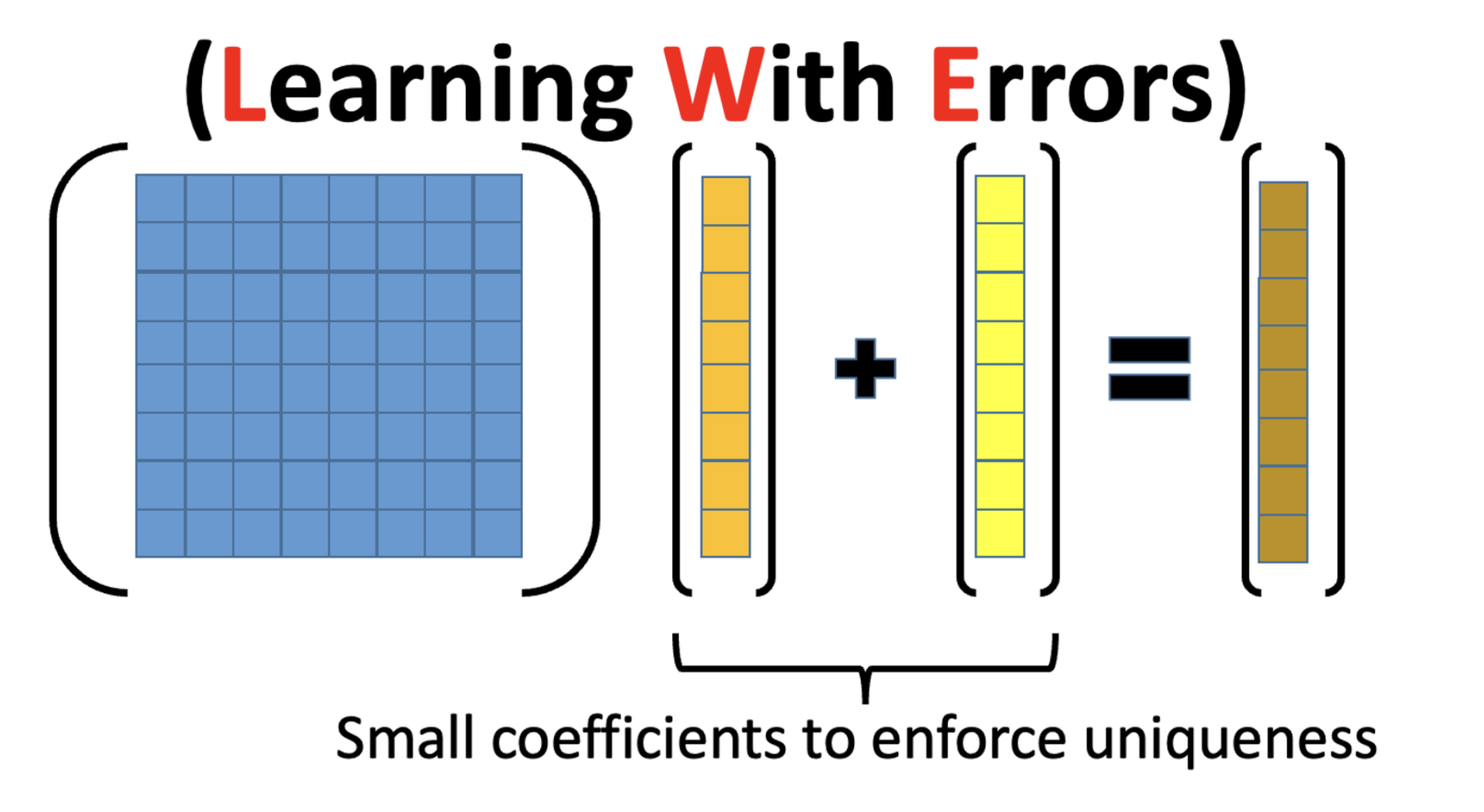
Grover's Algorithm can be used to speed up unstructured search and can be applied to finding symmetric keys and hash collisions.

Cryptography Today - Algorithms

RSA Encryption and Signatures,
(EC) Diffie-Hellman Key Exchange,
(EC) Digital Signature Algorithm,
(EC) ElGamal Encryption, Pairings.

Symmetric encryption like AES,
Hash functions like SHA2/3,
MAC schemes like HMAC.

Lattice-Based Cryptography



New Cryptographic Standards

FIPS 203

Federal Information Processing Standards Publication

Module-Lattice-Based Key-Encapsulation Mechanism Standard

Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

New Cryptographic Standards

FIPS 204

Federal Information Processing Standards Publication

Module-Lattice-Based Digital Signature Standard

Category: Computer Security **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

Challenges with PQC

Performance: larger ciphertexts and signatures, larger memory requirements, sometimes slower

Foundations: new assumptions, models, and analysis

Side-Channels: more research to study implementation

Variations: different use cases, combinations, different national and international standards, recommendations

Main Takeaways

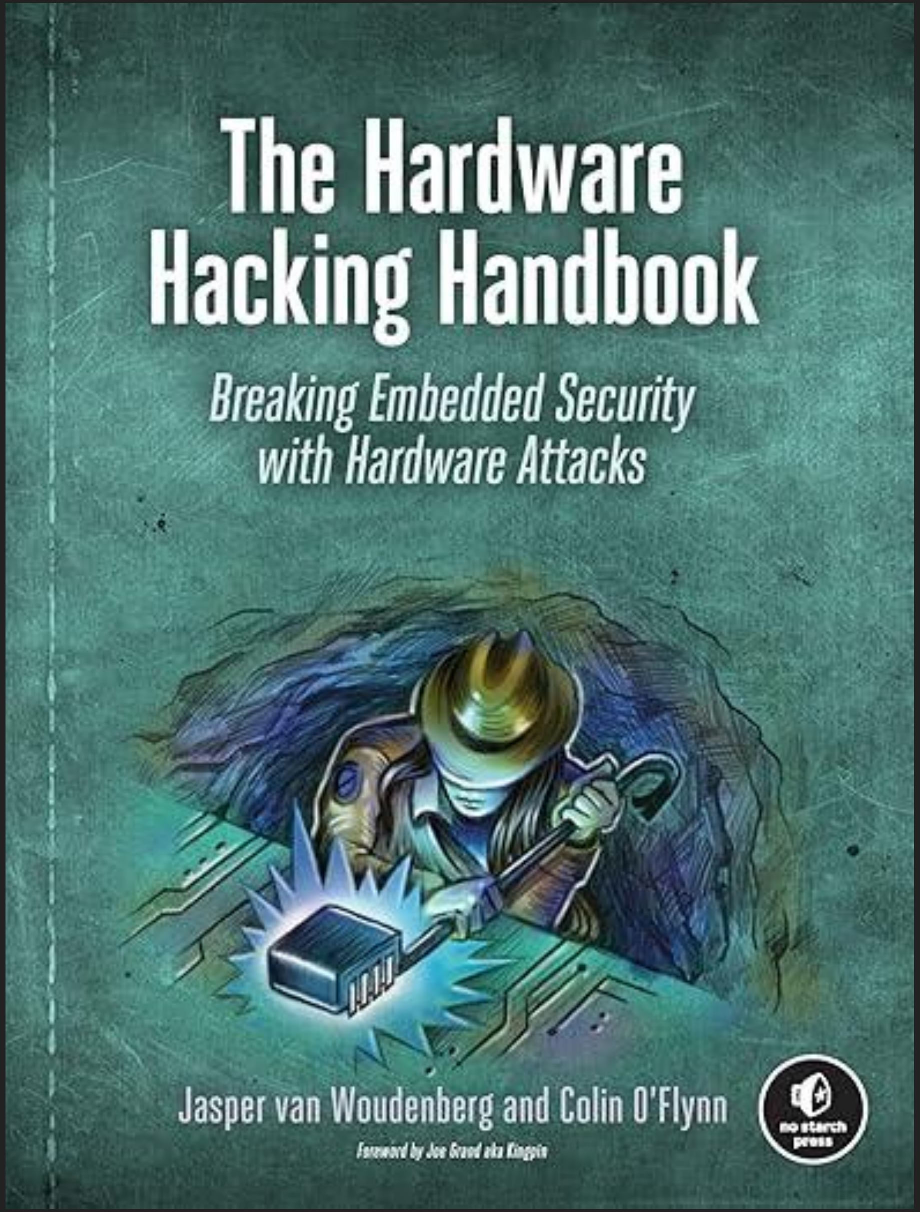
When it comes to crypto, correctness is not enough.

You should be aware of SCA and know what to look for.

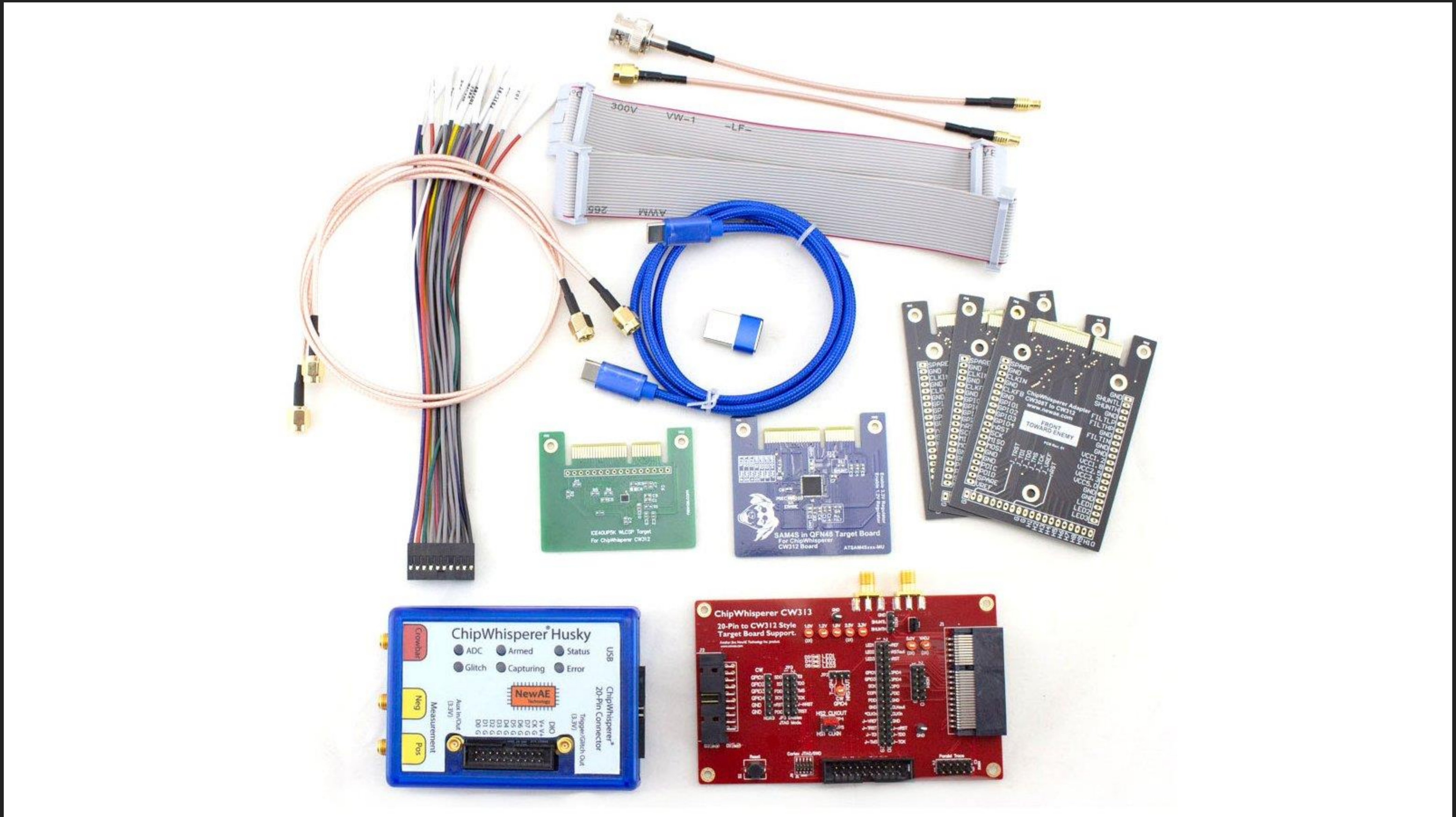
There exist many mitigation techniques for protection.

This will become a challenge going forward with PQC.

The Hardware Hacking Handbook



ChipWhisperer Husky



Thank you!
Questions?

Tjerand Silde, PONE Biometrics