Third Meeting

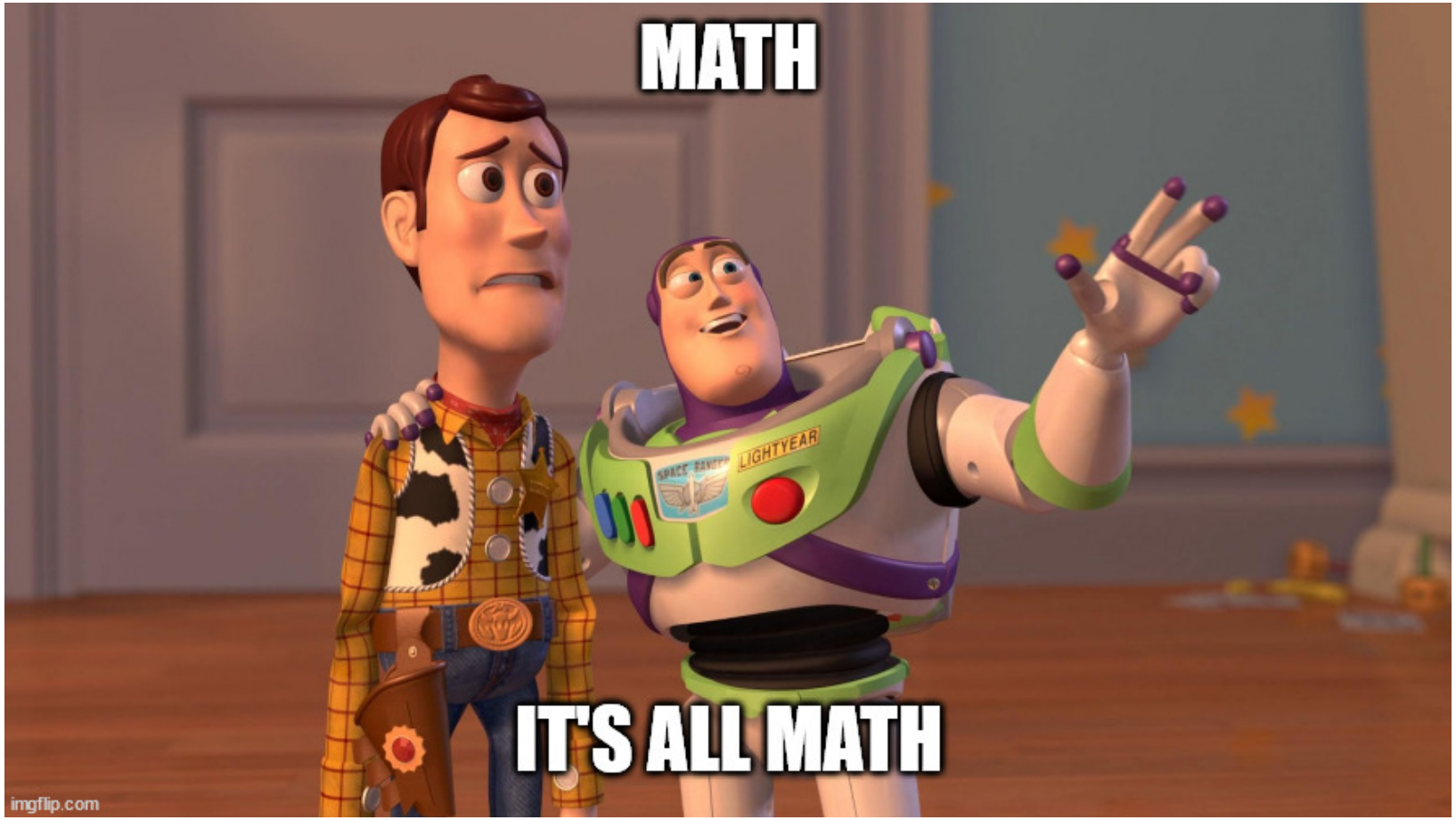
Public-Key Cryptography Standard: RSA

- RSA (Rivest-Shamir-Adleman)
- First published in 1977
- Documented in RFC 8017 (request for comment)
  - RFC is Technical documentations published by IETF (internet engineering task force)
  - Completely free and open for everyone to read (yay)

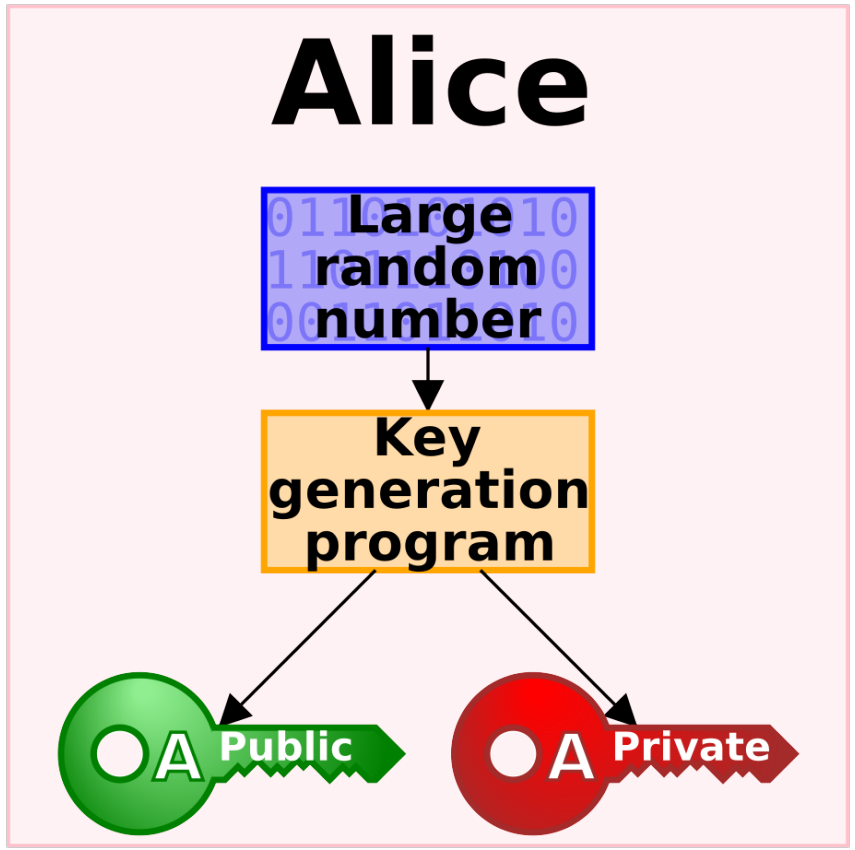
Public-Key Cryptography (PKC)

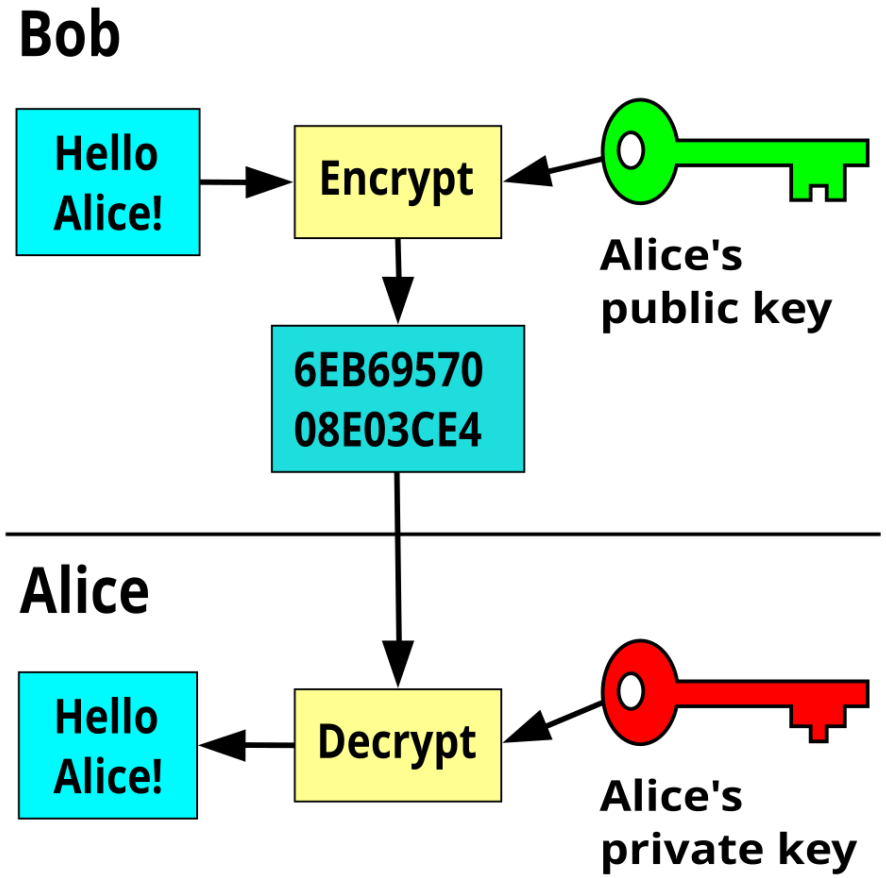
- **Asymmetric key**
  - Public key encrypt plaintext
  - Private key decrypt ciphertext
- Based on “hard to solve” problems, even to modern computers
- Integer Factorization Problem (IFP)
  - Finding a proper factor in a (large) number
- Discrete Logarithm Problem (DLP)
  - With $\{y, x, N\}$ known, $y = x^k (mod\ N) \xrightarrow{hard\ to\ find} k$

Cryptography?



Public-Key Cryptography





RSA Public-Key Cryptography (PKC)

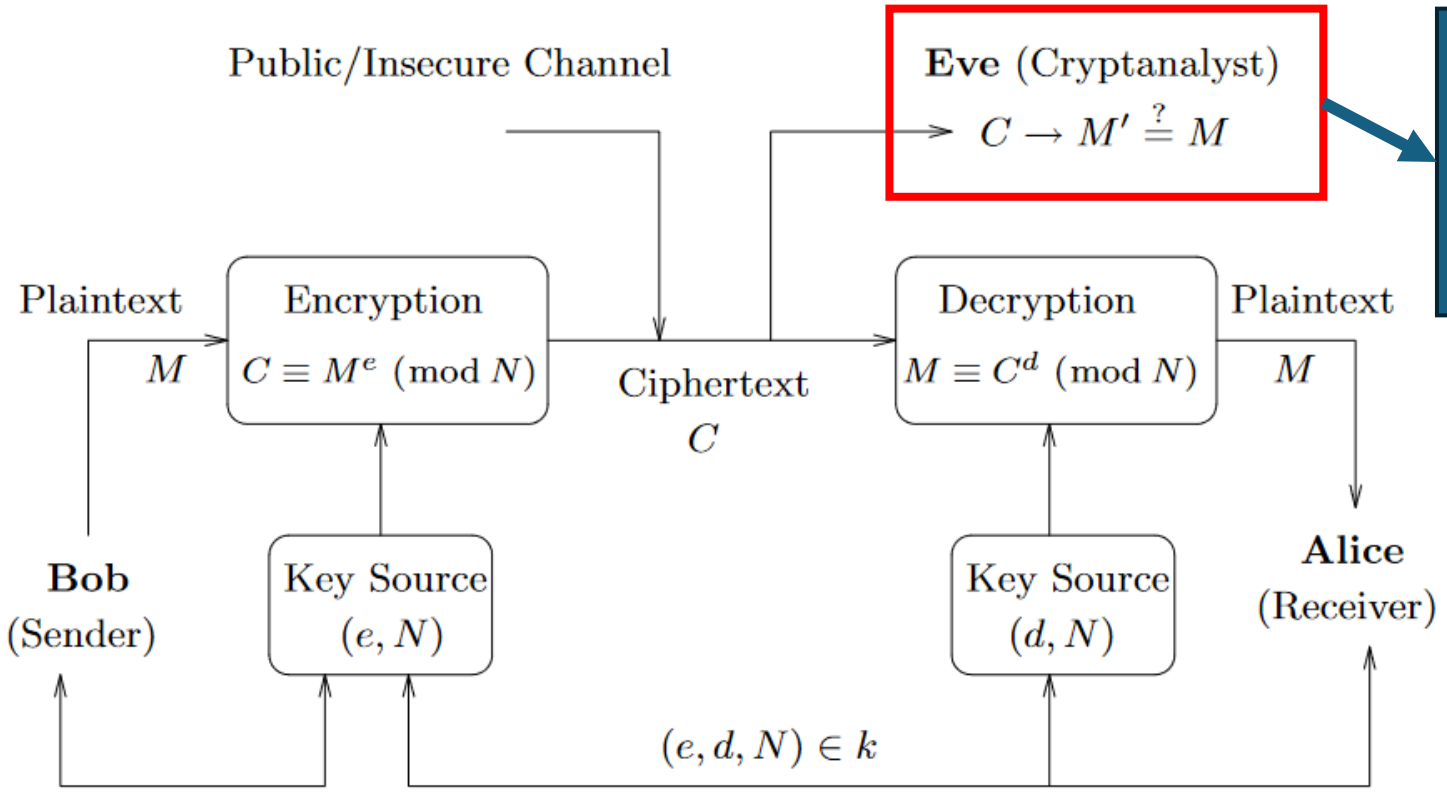


Figure 2.7. RSA Public-Key Cryptography

Because of the difficulty of IFP and DLP, it's difficult for Eve to retrieve the message without private exponent (d)

How does RSA work?

$$\text{RSA} = (\mathcal{M}, \mathcal{C}, \mathcal{K}, M, C, e, d, N, E, D) \tag{2.11}$$

where

(1) $\mathcal{M}$ is the set of plaintexts, called the plaintext space.

(2) $\mathcal{C}$ is the set of cipherexts, called the ciphertext space.

(3) $\mathcal{K}$ is the set of keys, called the key space.

(4) $M \in \mathcal{M}$ is a piece of particular plaintext.

(5) $C \in \mathcal{C}$ is a piece of particular ciphertext.

(6) $N = pq$ is the modulus with p, q prime numbers, usually each with at least 100 digits.

(7) $\{(e, N), (d, N)\} \in \mathcal{K}$ with $e \neq d$ are the encryption and **decryption** keys, respectively, satisfying

$$ed \equiv 1 \pmod{\phi(N)} \tag{2.12}$$

where $\phi(N) = (p-1)(q-1)$ is the Euler ϕ-function and defined by $\phi(N) = \#(\mathbb{Z}_N^*)$, the number of elements in the multiplicative group $\mathbb{Z}_N^*$.

- Euler totient function $\phi(N)$: how many integers from 1 to N are coprime to N
- N is the product of unique prime numbers

Encryption and Decryption

(8) E is the encryption function

$$E_{e,N}: \ M \mapsto C$$

That is, $M \in \mathcal{M}$ maps to $C \in \mathcal{C}$, using the public-key (e, N), such that

$$C \equiv M^e \pmod{N}. \tag{2.13}$$

(9) D is the decryption function

$$D_{d,N}: \ C \mapsto M$$

That is, $C \in \mathcal{C}$ maps to $M \in \mathcal{M}$, using the private-key (d, N), such that

$$M \equiv C^d \equiv (M^e)^d \pmod{N}. \tag{2.14}$$

RSA Proof

Proof. Notice first that

$$
\begin{aligned}
C^d &\equiv (M^e)^d \pmod N && \text{(since } C \equiv M^e \pmod N\text{)} \\
&\equiv M^{1+k\phi(N)} \pmod N && \text{(since } ed \equiv 1 \pmod{\phi(N)}\text{)} \\
&\equiv M \cdot M^{k\phi(N)} \pmod N \\
&\equiv M \cdot (M^{\phi(N)})^k \pmod N \\
&\equiv M \cdot (1)^k \pmod N && \text{(by Euler's Theorem } a^{\phi(n)} \equiv 1 \pmod N\text{)} \\
&\equiv M
\end{aligned}
$$

The result thus follows. □

Both encryption $C \equiv M^e \pmod N$ and decryption $M \equiv C^d \pmod N$ of RSA can be implemented in polynomial-time by Algorithm 1.3.5. For example the RSA encryption can be implemented as follows: p.53

- Proving that ciphertext (C) can be correctly decrypted by the private key(d, N) to get plaintext (M)
- Euler totient function ($\phi(N)$) makes it possible

RSA in short

- Secret: {d (private key), p , q, $\phi(N)$}
- Not secret: {e (public key), N (modulus), C (ciphertext)}
- Encryption: $C \equiv M^e \ (mod\ N)$
- Decryption: $M \equiv C^d \ (mod\ N)$
- Key pair: $ed \equiv 1 \left(mod\ \phi(N)\right)$
- Modulus: $N = pq$ with $p, q \in$ Primes

$$\{e, N, C \equiv M^e \ (\mathrm{mod}\ N)\} \xrightarrow{\text{hard}} \{M \equiv C^d \ (\mathrm{mod}\ N)\}. \qquad (2.26)$$

$$\{N, C \equiv M^e \ (\mathrm{mod}\ N)\} \xrightarrow[\text{easy}]{\{d,p,q,\phi(N)\}} \{M \equiv C^d \ (\mathrm{mod}\ N)\}. \qquad (2.28)$$

The idea of RSA

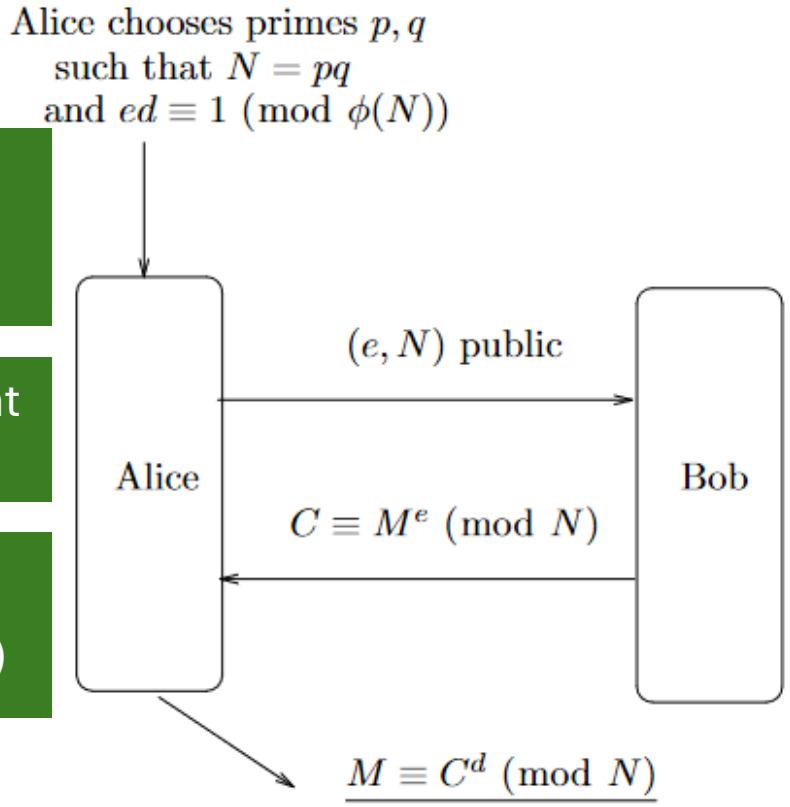


Figure 2.8. RSA Encryption and Decryption

Has the secrets

Gives out public exponent (e) and modulus (N)

Decrypts ciphers with the private exponent (d) and the modulus (N)

Does not have the secrets

Encrypts messages with the public exponent (e) and modulus (N)

Key points of RSA

- **Euler totient function $\phi(N)$:**
  - Generates the key pair
  - “Key” idea of asymmetric-key cryptography in RSA
- **Integer Factorization Problem:**
  - Makes factoring the modulus *N* itself difficult
- **Discrete Logarithm Problem:**
  - Makes extracting keys by inversing the encryption and decryption operations difficult

Other usage of PKC: Digital Signature

- Same operation as decryption and encryption
- **Signature signing (decryption):**
  - $S \equiv M^d (mod\ N)$
  - Can only be done by the authorized person who has the private exponent *d*
- **Signature verification (encryption):**
  - $M \equiv S^e (mod\ N)$
  - Can be done by anyone since (*e*, *N*) is public

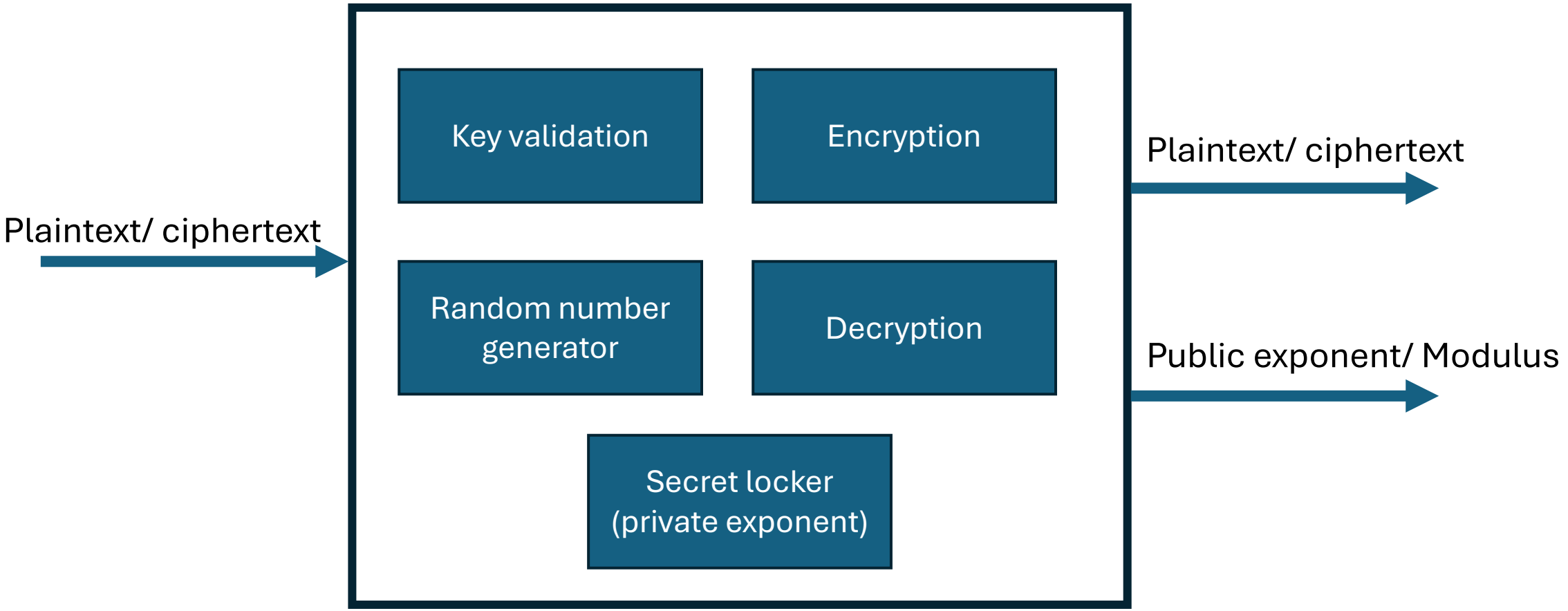
Problem with current Public-Key Cryptography

- **Quantum computer**
  - Can solve Integer Factorization Problem (IFP) and Discrete Logarithm Problem (DLP) in polynomial time
  - Most modern PKCs are based on IFP and DLP
  - Making most PKCs insecure
- Thus, here comes **Post-Quantum Cryptography**

Post-Quantum Cryptography (PQC)

- New PKC algorithms that are not based on IFP and DLP
- The development is lead by NIST (national institute of standards and technology, of the U.S. department of commerce)
- NIST just published (2024.8.13) the final versions of PQCs
  - FIPS 203, 204, 205
  - Can be implemented now
  - (I’m reading them but they all look funny and hard to understand)

Cryptographic Hardware Accelerator



Paper that I'm still reading...

2018 IEEE Symposium on Security and Privacy

FPGA-Based Remote Power Side-Channel Attacks

Mark Zhao and G. Edward Suh
Computer Systems Laboratory
Cornell University
Ithaca, New York
yz424@cornell.edu, suh@ece.cornell.edu

what even is color code

***Abstract*—The rapid adoption of heterogeneous computing has driven the integration of Field Programmable Gate Arrays (FPGAs) into cloud datacenters and flexible System-on-Chips (SoCs). This paper shows that the integrated FPGA introduces a new security vulnerability by enabling software-based power side-channel attacks without physical proximity to a target system. We first demonstrate that an on-chip power monitor can be built on a modern FPGA using ring oscillators (ROs), and characterize its ability to observe the power consumption of other modules on the FPGA or the SoC. Then, we show that the RO-based FPGA power monitor can be used for a successful power analysis attack on an RSA cryptomodule on the same FPGA. Additionally, we show that the FPGA-based power monitor can observe the power consumption of a CPU on the same SoC, and demonstrate that the FPGA-to-CPU power side-channel attack can break timing-channel protection for an RSA program running on a CPU. This work introduces and demonstrates remote power side-channel attacks using an FPGA, showing that the common assumption that power side-channel attacks require specialized equipment and physical access to the victim hardware is not true for systems with an integrated FPGA.**

I. Introduction

As we increasingly rely on hardware acceleration to improve the performance and energy efficiency of computing systems, Field Programmable Gate Arrays (FPGAs) have recently been widely adopted in large-scale datacenters. For example, Amazon offers FPGA instances in its EC2 service, allowing customers to rent FPGAs in its cloud computing environment [1]. Microsoft heavily utilizes FPGAs in its datacenters for various tasks ranging from web searches to network crypto and machine learning [2]. Similarly, Baidu measure the power consumption as the voltage drop across the resistor. In this paper, we demonstrate that an on-chip power monitor can be constructed using the programmable logic of an FPGA, allowing us to measure dynamic power consumption with sufficient resolution to enable power analysis attacks. In essence, the integrated FPGA opens the door for remote power analysis attacks.

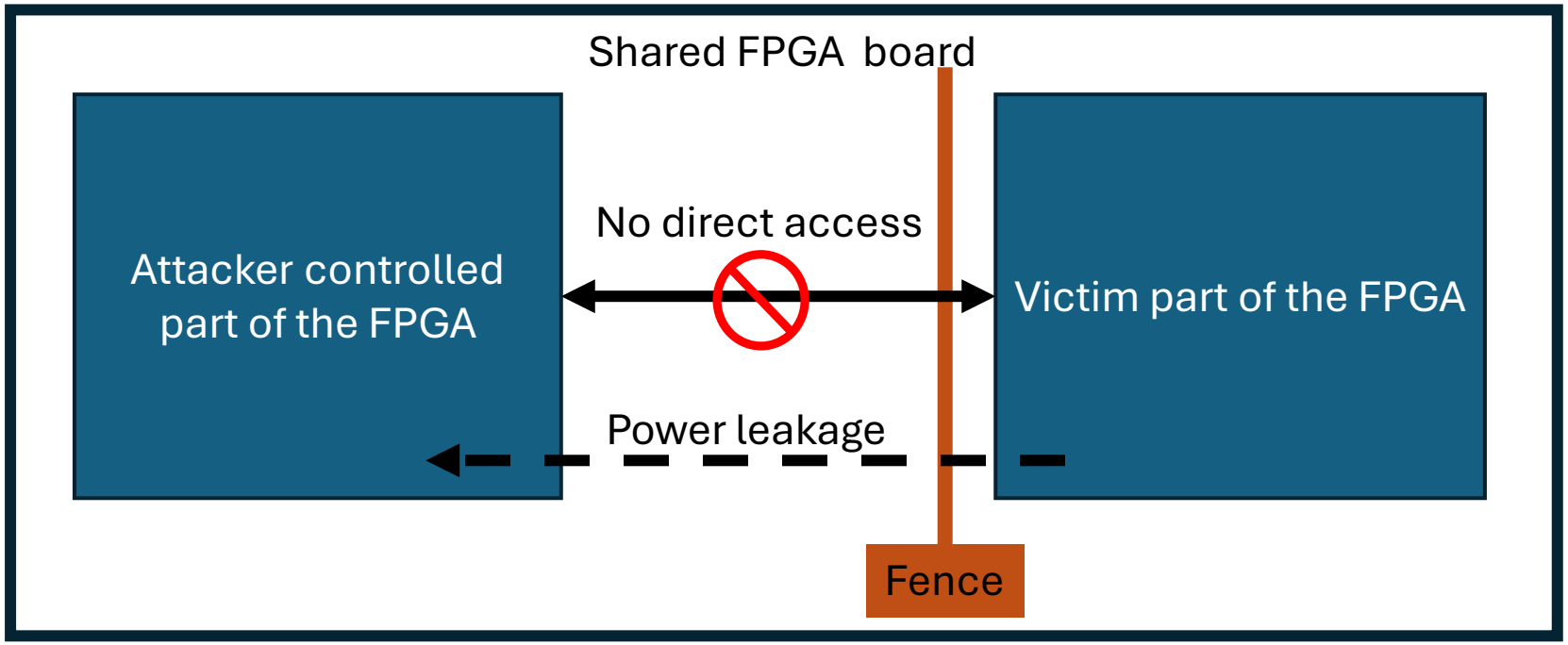
This FPGA-based power side channel may be exploited in a variety of system architectures that allows an untrusted user to program a part of an FPGA. In cloud computing infrastructures, many studies from both academia and industry have proposed mechanisms to virtualize and share FPGAs among multiple users so that multiple accelerators co-reside on one physical FPGA [5]–[10]. Even in cloud platforms where each FPGA is allocated to a single user, untrusted user logic is co-resident with privileged control logic called the 'shell' [11]. Similarly, in personal computing platforms, an FPGA fabric can be shared among multiple programs, including a potentially malicious application. In such a shared FPGA platform, we show that an FPGA-to-FPGA attack, where an attack circuit in one part of the FPGA steals a secret used by another circuit on the same FPGA, is viable.

vector

As a concrete example, we demonstrate a simple power analysis (SPA) attack on an RSA accelerator on an FPGA. In this example, we implement an RSA decryption engine and a power monitor on one FPGA, but isolate both logically and physically; there is no connection between the two modules and they are implemented at physically different locations on the FPGA. This is analogous to either two users or one

FPGA-Based Remote Power Side-Channel Attacks

- The author proposed that it's possible to build a on-chip power monitor using ring oscillators, using HDL, and perform remote power side-channel attacks



Thanks for listening

:P