THE GRAINGER COLLEGE OF ENGINEERING

CS 521

Technological Foundations of Blockchain and Cryptocurrency

Grigore Rosu

Topic 2 – Basic Crypto Primitives

ILLINOIS

Thanks!

To Professors

David Tse (Stanford)

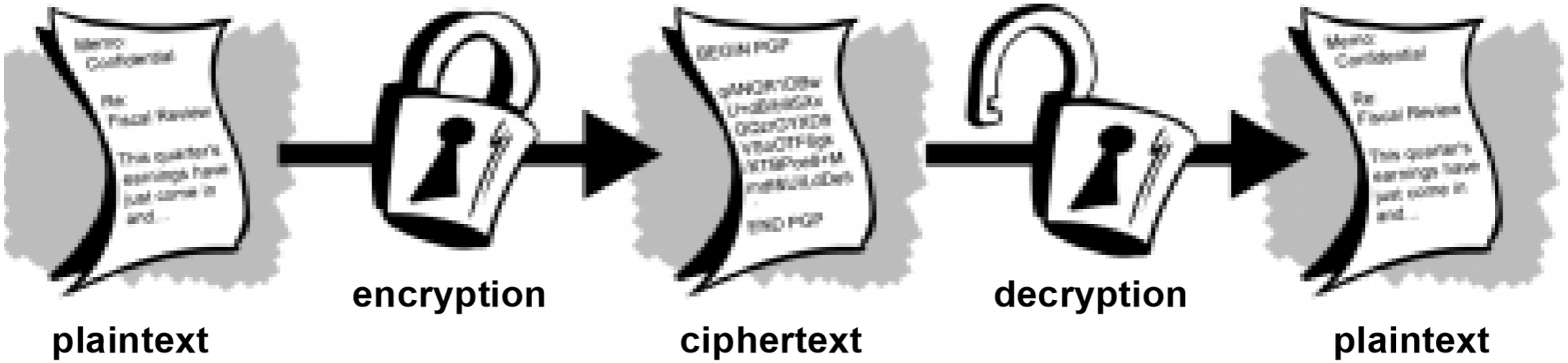
Sriram Viswanath (UT Austin)

Sreeram Kannan (UW – now at EigenLayer)

Some crypto primitives

- Encryption and Signatures
- Cryptographic Hash Functions
- Hash Accumulators
  - Blockchain
  - Merkle trees

Basic Encryption



hello $\xrightarrow{\text{Encryption}}$ lipps $\xrightarrow{\text{Decryption}}$ hello

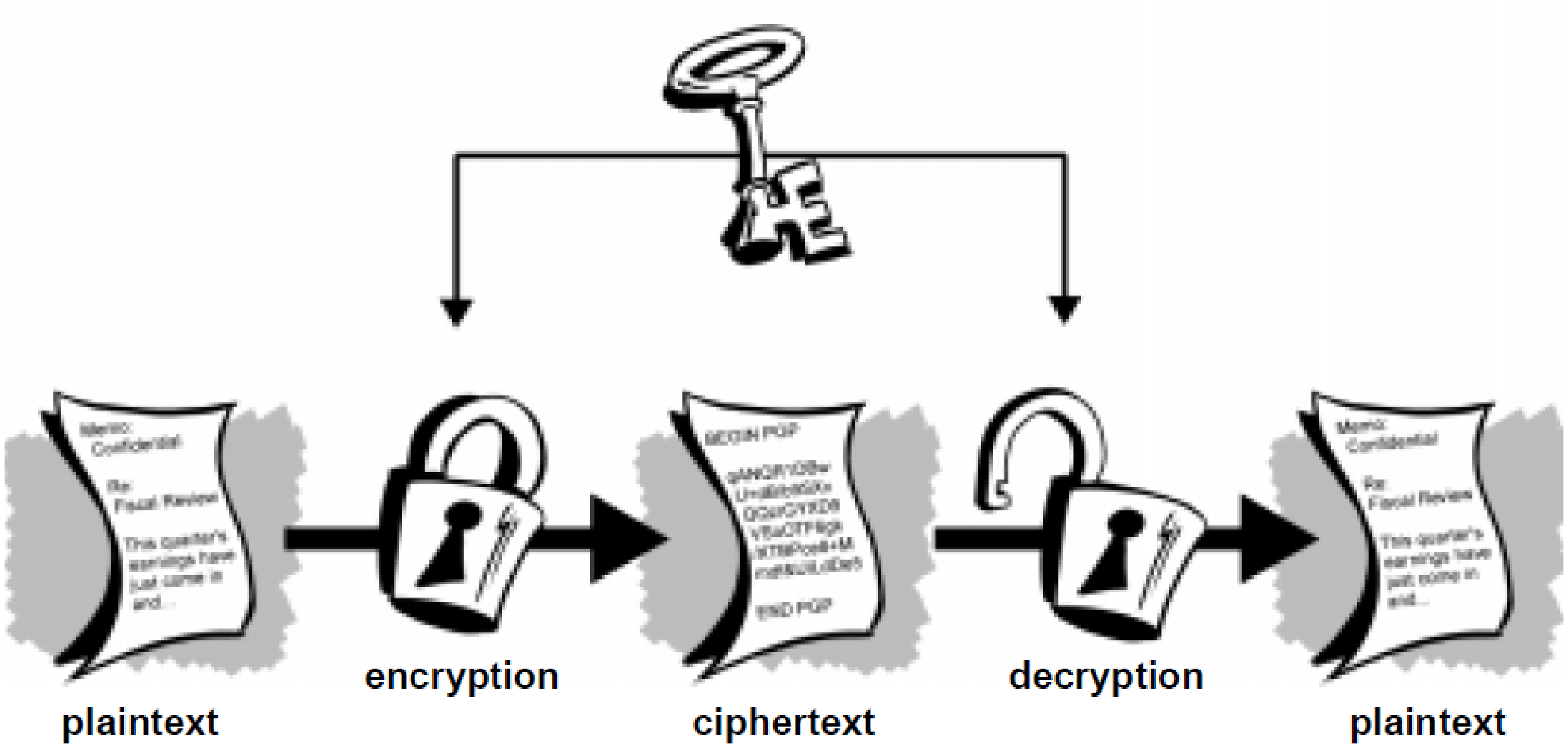
plaintext *ciphertext* *plaintext*

Cypher: Offset the Alphabet
Key: 4

Scene from “Breaking the Enigma Code”

https://youtu.be/zZuqLLdx2YQ

Symmetric (aka Secret-Key) Cryptography



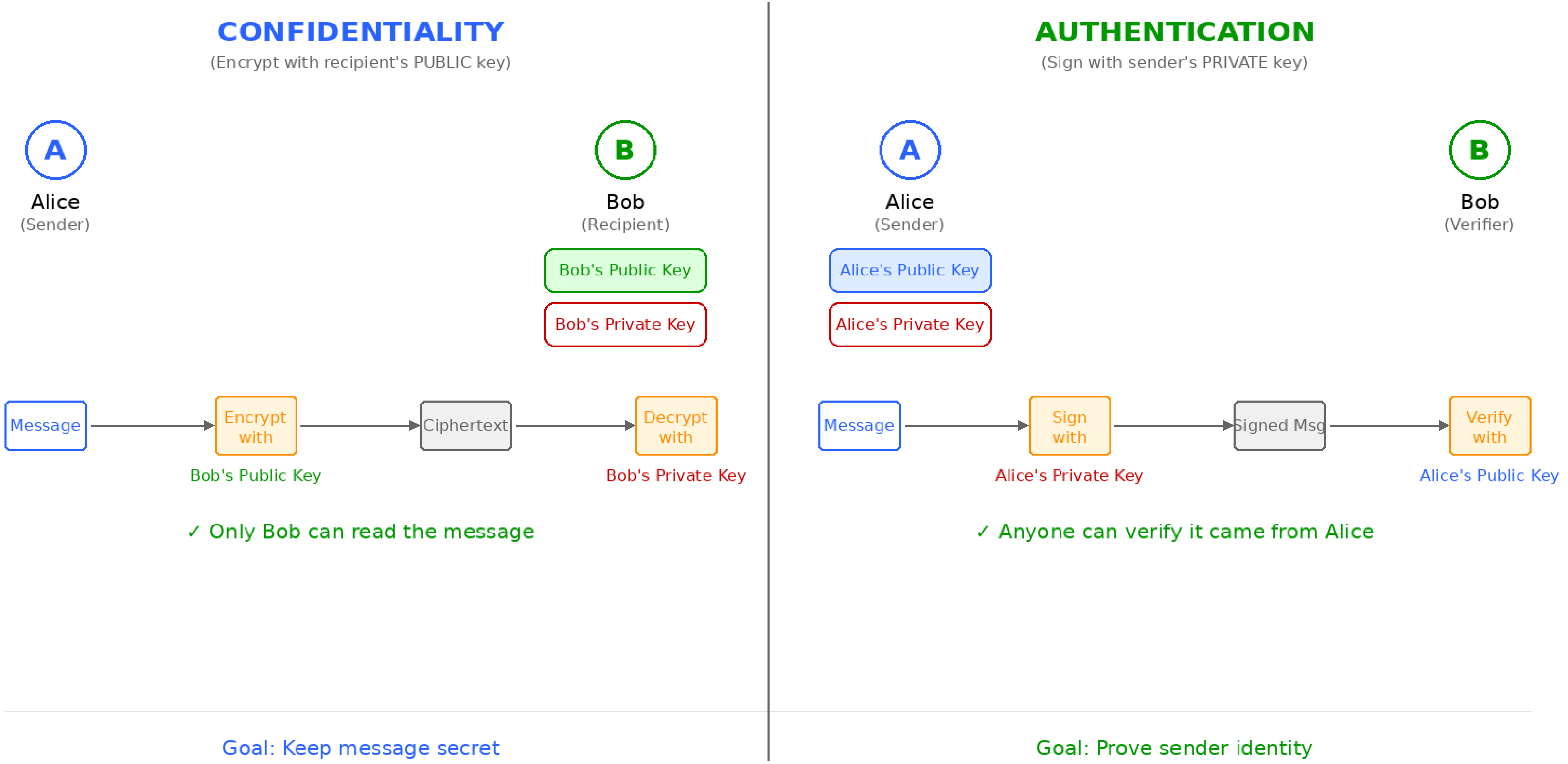
How Secret-Key Cryptography Works

- **Single Shared Key** – Both sender and receiver use the same secret key for encryption and decryption
- **Key Distribution** – The shared key must be securely exchanged between parties before communication
- **Fast Performance** – Symmetric algorithms are computationally efficient, ideal for encrypting large amounts of data
- **Common Algorithms** – Examples include AES (Advanced Encryption Standard), DES, and 3DES

Pros and Cons of Secret-Key Cryptography

- ✓ High performing – fast, especially if the data is not going to be transmitted
- ✓ Can be implemented in hardware and software
- ✗ Secure key distribution is difficult, requires trust and secrecy between the parties as well as trust for the “distribution mechanism” if the parties are not in the same location

Asymmetric (aka Public-Key) Cryptography



How Public-Key Cryptography Works

- **Key Pair** – Each party has a public key (shared openly) and a private key (kept secret)
- **No Key Exchange Required** – Public keys can be freely distributed; only the private key must remain secret
- **Asymmetric Encryption** – Data encrypted with one key can only be decrypted with the other key in the pair
  - **Confidentiality** – Encrypt with recipient's public key; only they can decrypt with their private key
  - **Authentication** – Encrypt with your private key; anyone can verify it came from you using your public key
- **Common Algorithms** – Examples include RSA, Elliptic Curve Cryptography (ECC), and Diffie-Hellman

RSA Cryptosystem

1. Key Generation

Choose large primes $\boxed{p}$ and $\boxed{q}$

typically 1024+ bits each

Compute $n = p \times q$

n is public, 2048+ bits

Compute $\boxed{\varphi(n)} = (p-1)(q-1)$

Euler's totient = count of integers < n coprime to n

Choose e with $1 < e < \varphi(n)$, $\gcd(e, \varphi(n)) = 1$

Common $e = 65537$ (Fermat prime $2^{16}+1$)

Compute $\boxed{d} = e^{-1} \bmod \varphi(n)$

Extended Euclidean Algorithm: since $\gcd(e, \varphi(n)) = 1$,
Bézout's identity guarantees $\exists\, d, k$: $e \cdot d + \varphi(n) \cdot k = 1$
Repeat division: $r_0 = \varphi(n)$, $r_1 = e$, $r_{i+1} = r_{i-1} \bmod r_i$ until $r_k = 0$
Back-substitute to find d from the quotients
$O(\log n)$ steps — very fast even for 2048-bit numbers

Public Key	Private Key
(n, e)	**(n, d)**
share openly	keep secret

2. Confidentiality

Encrypt (public key)

Message m
↓
$c = m^e \bmod n$
↓
Ciphertext c

→

Decrypt (private key)

Ciphertext c
↓
$m = c^d \bmod n$
↓
Message m

3. Authentication

Sign (private key)

Message m
↓
$s = m^d \bmod n$
↓
Signature s

→

Verify (public key)

Signature s
↓
$m = s^e \bmod n$
↓
Message m ✓

Why RSA Works

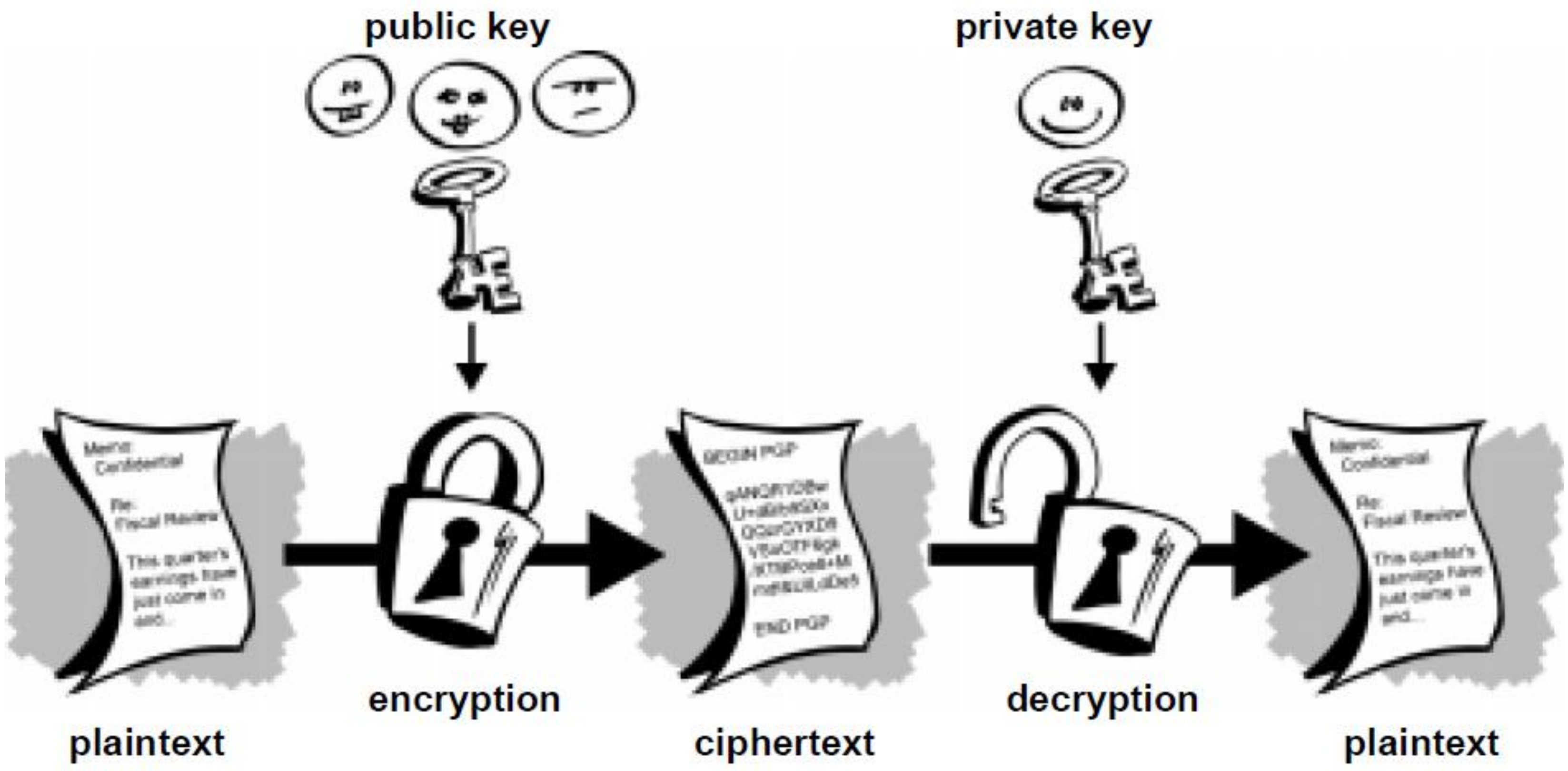
Security: Factoring $n = p \times q$ is computationally infeasible for large primes

Euler's Theorem: If $\gcd(m, n) = 1$, then $m^{\varphi(n)} \equiv 1 \pmod n$

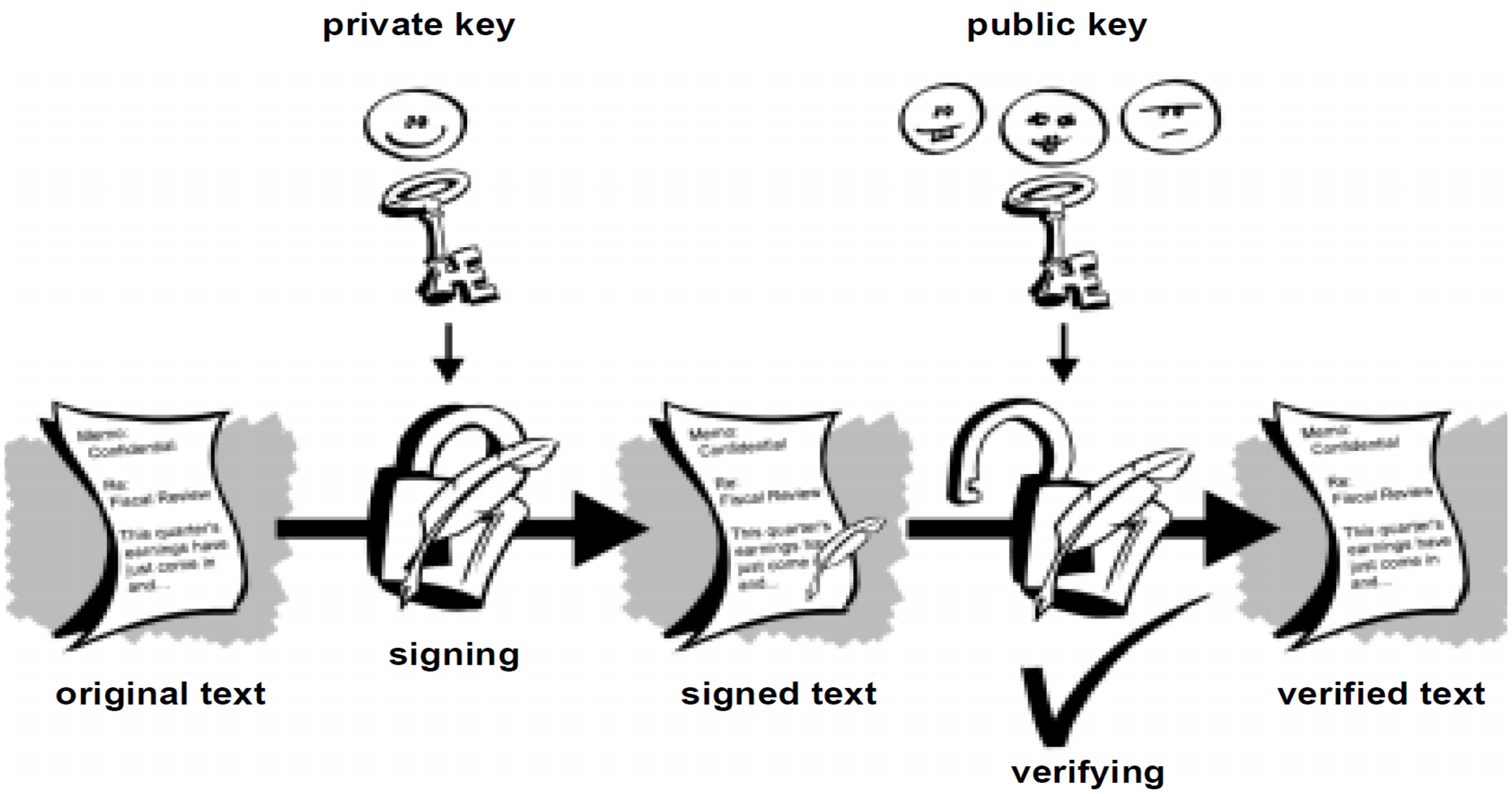
Confidentiality: $e \cdot d \equiv 1 \pmod{\varphi(n)} \Longrightarrow c^d = (m^e)^d = m^{(e \cdot d)} = m \cdot (m^{\varphi(n)})^k \equiv m \pmod n$

Authenticity: $s^e = (m^d)^e = m^{(d \cdot e)} \equiv m \pmod n$; only private key holder can produce valid s

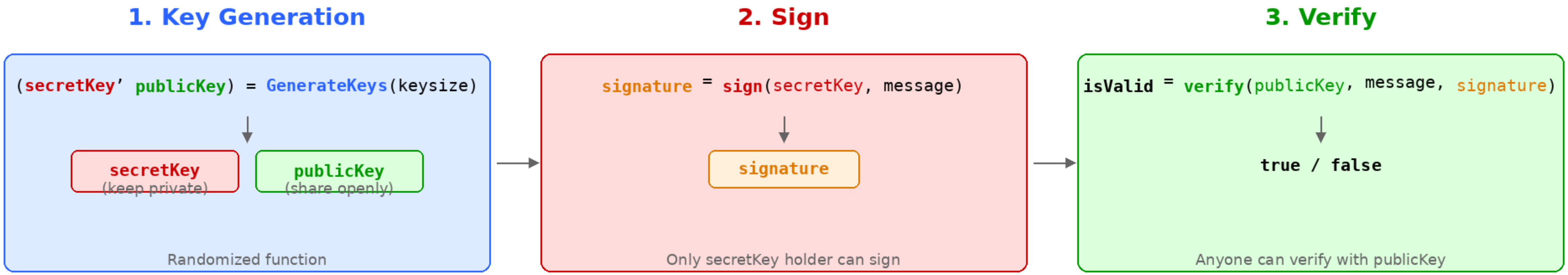
Confidentiality – Encrypt with Recipient's Public Key



Authentication – Digital Signatures



Digital Signatures API



Key Properties

Security Guarantees:

- Authenticity: Only the secret key holder can create valid signatures
- Integrity: Any modification to the message invalidates the signature
- Non-repudiation: Signer cannot deny having signed the message

Common Algorithms:

- RSA-PSS: RSA with probabilistic padding
- ECDSA: Elliptic Curve Digital Signature Algorithm
- EdDSA: Edwards-curve DSA (Ed25519)

Unforgeable Signatures

- **Existential Unforgeability** – Computationally infeasible to forge a valid signature on any message without the secret key
- **Adaptive Chosen Message Attack** – Secure even if adversary can request signatures on chosen messages before attempting forgery
- **Computational Hardness** – Based on hard math problems: discrete log (ECDSA), integer factorization (RSA)
- **Common Algorithms**
  - **ECDSA** – Elliptic Curve Digital Signature Algorithm; used in Bitcoin and Ethereum
  - **EdDSA** – Edwards-curve DSA (Ed25519); faster and used in newer protocols
  - **Schnorr** – Provably secure with signature aggregation; adopted by Bitcoin (Taproot)

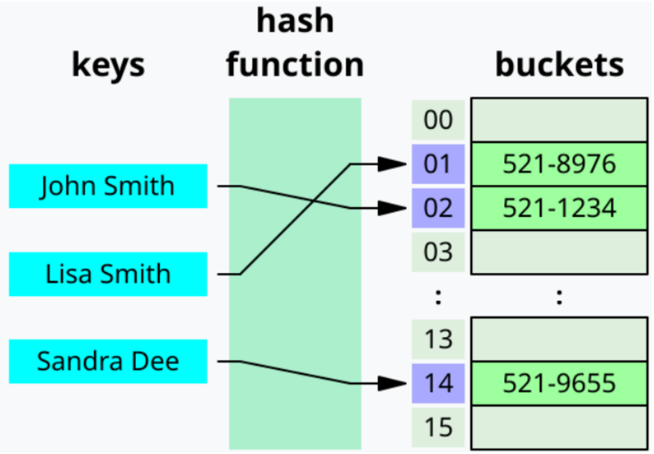
Pros and Cons of Public-Key Cryptography

- ✓ **No Pre-Shared Secret** – Communicate securely without prior key exchange arrangement
- ✓ **Scalable Key Management** – N users need only N key pairs (vs N^2 keys for symmetric)
- ✓ **Digital Signatures** – Enables authentication, integrity, and non-repudiation
- ✓ **Decentralized Identity** – Public keys can serve as pseudonymous identities (addresses)
- ① **Computationally Expensive** – 100-1000x slower than symmetric encryption
- ① **Key-Identity Binding** – Public key alone doesn’t prove real-world identity of holder
- ① **Secret Key Protection** – Loss or theft of private key compromises all security
- ① **Quantum Vulnerability** – RSA/ECDSA broken by quantum computers (Shor’s algorithm)

Decentralized Identity Management

- Public keys are your identity
  - *address* in Bitcoin/blockchain terminology
- Can create multiple identities
  - (`publickey`, `secretkey`) pairs
  - Publish `publickey`
  - Sign using `secretkey`
- Can create oneself
- Verifiable by others

Hash Functions



Defining Properties:

- Arbitrary sized inputs
- Fixed size deterministic output
- Efficiently computable
- Minimize collisions

Canonical application:

- Hash Tables
- Store and retrieve data records

Example: Hash Functions

- Division hashing

$$y = x \mod 2^{256}$$

- Uniform output
- Simple deterministic function
- Collision resistant

Cryptographic Hash Functions

Extra Properties:

- Adversarial collision resistance
  - Birthday paradox
- One way function
- Specialized one way function

Canonical applications:

- Message digest
- Commitments
- Puzzle generation
- Mining process

Hashing Algorithms

NSA 2001

No Collisions (yet)

SHA2 (Secure Hashing Algorithm)

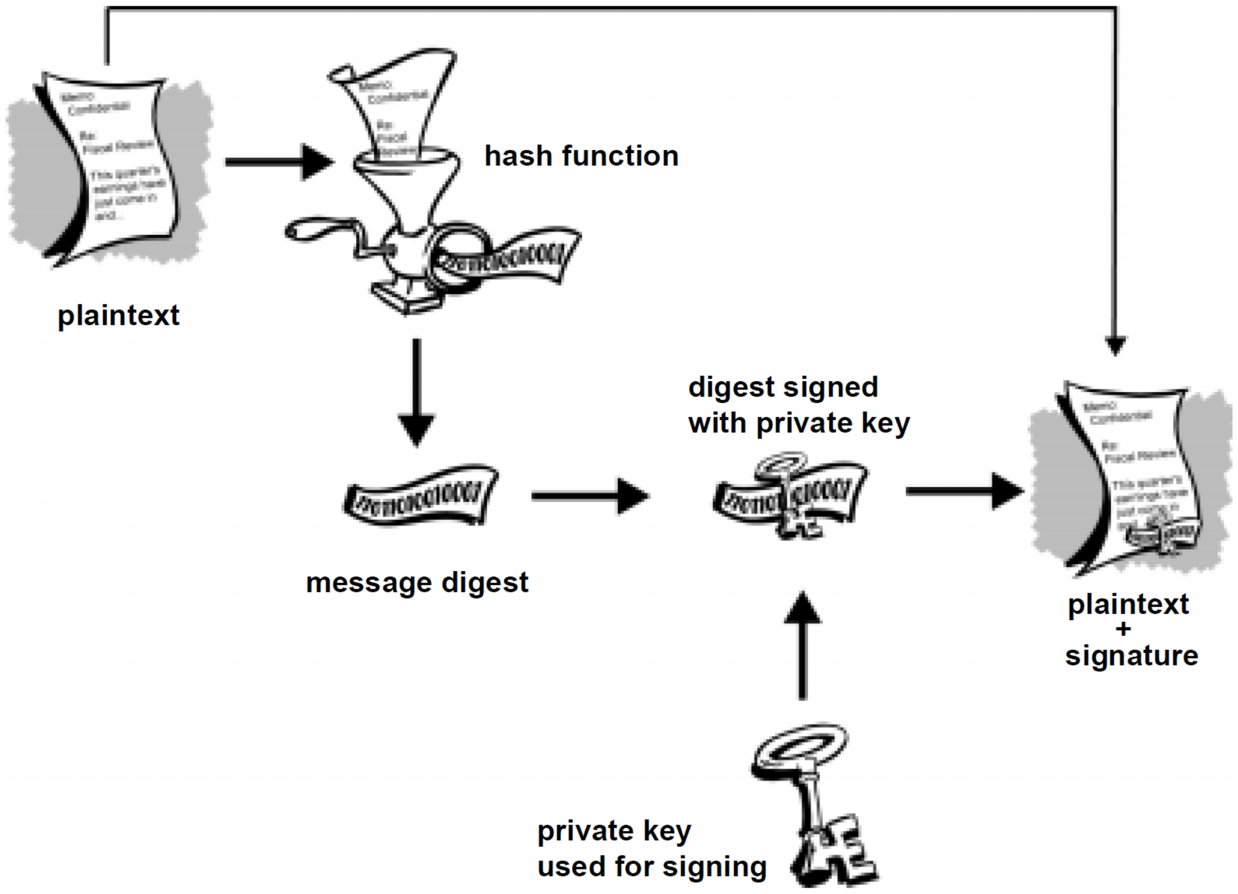
- SHA2 takes strings of arbitrary length and generates a unique and irreversible 256 (SHA256) or 512 (SHA512) bit strings (SHA2 is the successor to SHA1 that generated 160 bit strings)
- SHA1 was derived from MD4

MD5 (Message Digest)

- MD5 is also a “child” of MD4 and produces a 128 bit output string
- MD5 works by chaining a “compression function

Collisions found!

Basic building blocks together



Hash Pointer

- **Pointer to:**

  location of information

  + hash of the information
- **Regular pointer**
  - retrieve information
- **Hash pointer**
  - retrieve information and verify the information has not changed

- Regular pointers
  - Used to build data structures
    - linked lists, binary trees, etc
- Hash pointers
  - Can also be used to build data structures
  - Crucially useful for blockchains!
    - Blockchain = hash pointer based data structure

Blockchain: a linked list via hash pointers

- **Block**: Header + Data
- **Header:** hash pointer to

  location of previous block

  \+ hash of the previous block
- **Data**: information specific to the block (e.g., transactions)

- **Application**: tamper evident information log
- Head of the chain being known is enough to find tamper evidence in any internal block
- Hence the phrase: **block chain**

  blockchain

Merkle Tree

Binary tree of hash pointers

- Retain only the tree root
- Tamper of any data in the bottom of the tree is evident

- **Proof of Membership**
- **Proof of Non-membership**

Merkle Trees

- **Block**: Header + Data
- **Header:** Pointer to
  location of previous block
  + hash of the previous block
- **Data**
  – block specific information