Cryptography

Peter Chapin

Vermont Technical College

CIS-3720

Encryption Modes

- Encryption algorithms can be used several different ways (modes)
- Each mode has relative pros and cons
- Mode behavior independent of the underlying algorithm
  - All algorithms can be used in several modes
  - Strengths and weakness of a mode are specific to the mode, not the algorithm
- When communicating…
  - Partners must use the same algorithm (assumed known to attacker)
  - Partners must use the same mode (assumed known to attacker)
  - Partners must use the same key (**unknown to attacker**)

Mode Issue: Error Propagation

- *If a single bit error occurs in the cipher text, how much plain text is trashed?*
  - **At least one block will be trashed**; a good cipher has high diffusion
  - Ideally only one block will be trashed
  - Some modes might trash everything after the error

Mode Issue: Random Access

- If it is necessary to modify a random bit of plain text, how much cipher text needs to be decrypted and re-encrypted?
  - Ideally only one block should need to be processed
  - Some modes might require the decryption and re-encryption of all data after the modification…
  - … or before the modification…
  - … or both

Mode Issue: Block Cipher Decryption?

- Does the mode require the block cipher to be used as a decryptor?
  - Some modes use the block cipher as an encryptor even when decrypting
  - Advantage: If decryption is slow, you don't have to worry about that (for example, AES decryption is slower than encryption)
  - Advantage: Encryption can be one-way without any feasible way to decrypt

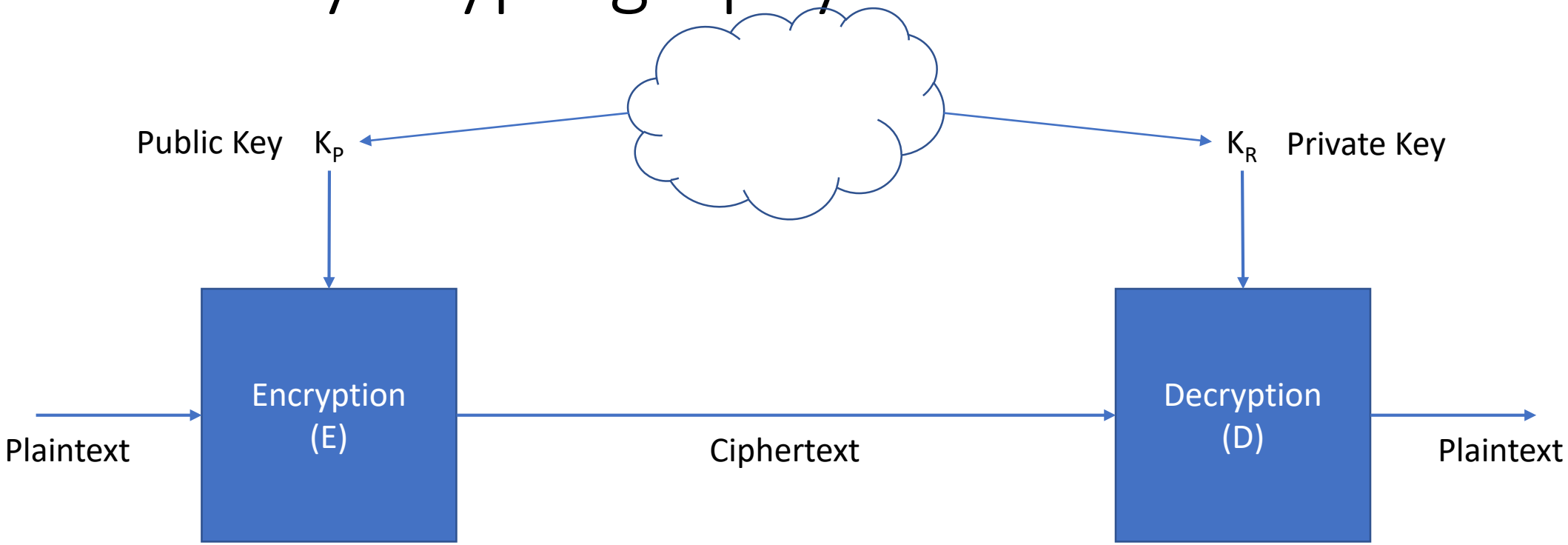
Mode Issue: Initialization Vector

- Does the mode need an IV and, if so, how should it be handled?
- Disadvantages…
  - IV is another piece of data that needs to be shared between partners
  - IV increases the size of the ciphertext beyond the size of the plain text
  - IV probably needs to be a *nonce*
    - In some modes the same IV (and key) will lead to the same key stream. If two plaintexts are encrypted with the same key stream, XORing the cipher texts together makes extracting both plain texts easy.
  - Generating quality nonces can be hard

Modes…

- ECB (Electronic Code Book mode)
  - Easy to implement, good error propagation, easy random access; identical plain text encrypts to identical ciphertext
- CBC (Cipher Block Chaining)
  - Difficult random access
- CFB (Cipher Feedback)
- OFB (Output Feedback)
  - Essential to never use the same IV with the same key or else security is destroyed.

Public Key Cryptography



Public Key Encryption: $K_P \neq K_R$

Basic Use Case

- The steps…
  1. Alice computes (public, private) key-pair.
  2. Alice posts public key anywhere (e. g., her web site)
  3. Alice keeps private key secure
  4. Bob downloads Alice's public key
  5. Bob encrypts a message for Alice's eyes only
  6. Bob sends message to Alice
  7. Alice uses private key to decrypt message
- *Not feasible for attacker to compute private key from public key*

A Few Observations

- Public key algorithms are (usually) slower than symmetric algorithms
- Public key algorithms are based on mathematically “hard” problems, not “confusion” and “diffusion”
- Key lengths can’t be compared with symmetric algorithms
- Public key algorithms are not necessary more secure than symmetric algorithms
- Public key algorithms compliment, but do not replace symmetric algorithms
  - **Good for symmetric key distribution; not good for bulk data transfer**

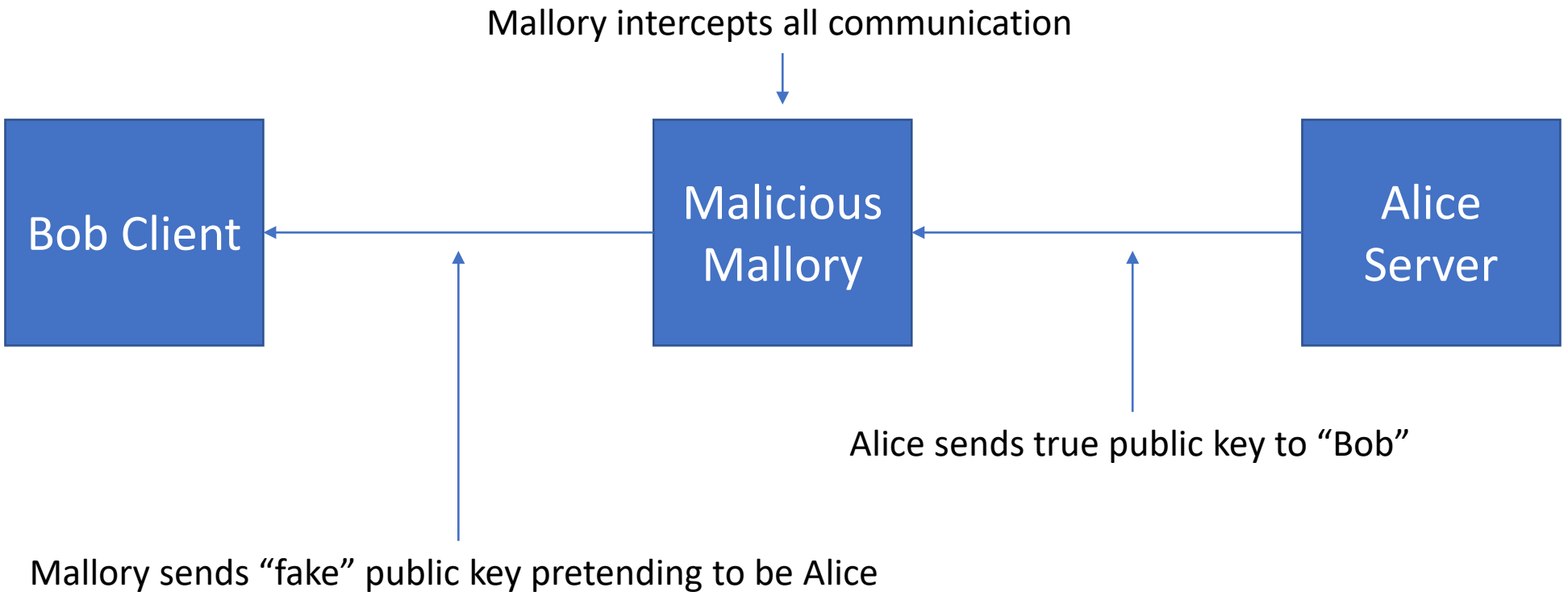
Key Exchange

- Public key algorithms solve the key exchange problem
  - SERVER: Sends its public key to the client. Doesn't matter if seen on network
  - CLIENT: Generates a random session key for a symmetric algorithm
  - CLIENT: Encrypts session key with server's public key
  - CLIENT: Sends encrypted session key to server
  - SERVER: Decrypts session key with private key
- Client & Server now share a symmetric key
  - All subsequent communication (bulk data transfer) encrypted with a symmetric algorithm. *Faster!*
  - Eve can't decrypt session key (can't compute server's private key)

Digital Signatures

- The steps…
  - Alice encrypts message with her private key
  - Alice posts message on her web site (or sends to Bob)
  - Bob uses Alice's public key to decrypt message
    - If successful, Bob knows only Alice could have encrypted the message
- What is "success" in this context?
  - There's a bit more involved
  - … must wait until we talk about cryptographic hashes

New Problem: (Wo)Man-in-the-Middle



Mallory negotiates (separate) session keys with Alice and Bob; can now read and modify all messages!

Trusted Third Party

- The steps…
  - Alice generates a (public, private) key-pair
  - Alice submits her public key to Trent, a trusted third party
  - Trent authenticates Alice
  - Trent digitally signs Alice's public key: creates a *certificate*
  - Alice posts her public key certificate on her web site
- Now…
  - Bob download's Alice's certificate
  - Bob verifies Trent's digital signature on the certificate
  - Bob believes he has Alice's true key

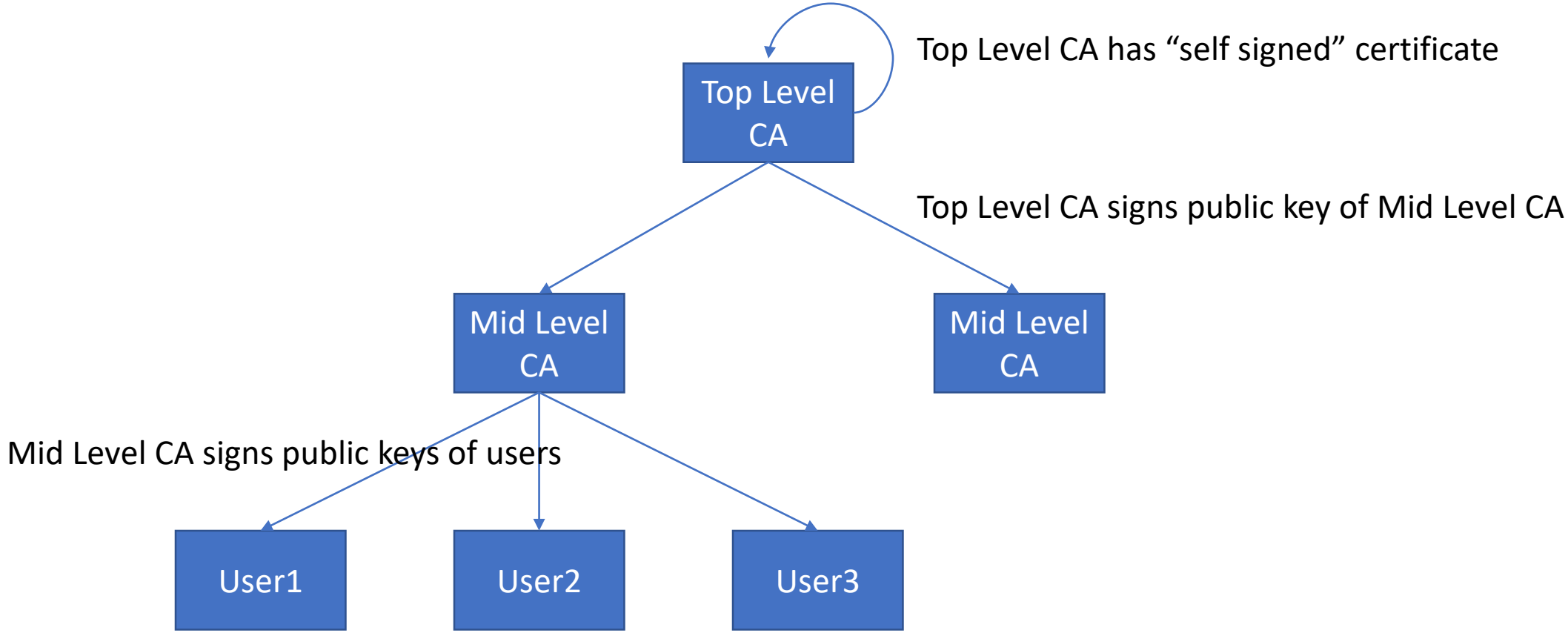
What's Mallory To Do?

- The steps…
  - Mallory creates a (public, private) key-pair
  - Mallory presents public key to Trent as "Alice"
  - Trent requires Mallory prove she is Alice
  - Mallory fails to do so. No certificate is made
- *Assumes: Mallory can't trick Trent*. **Trent must know his business**

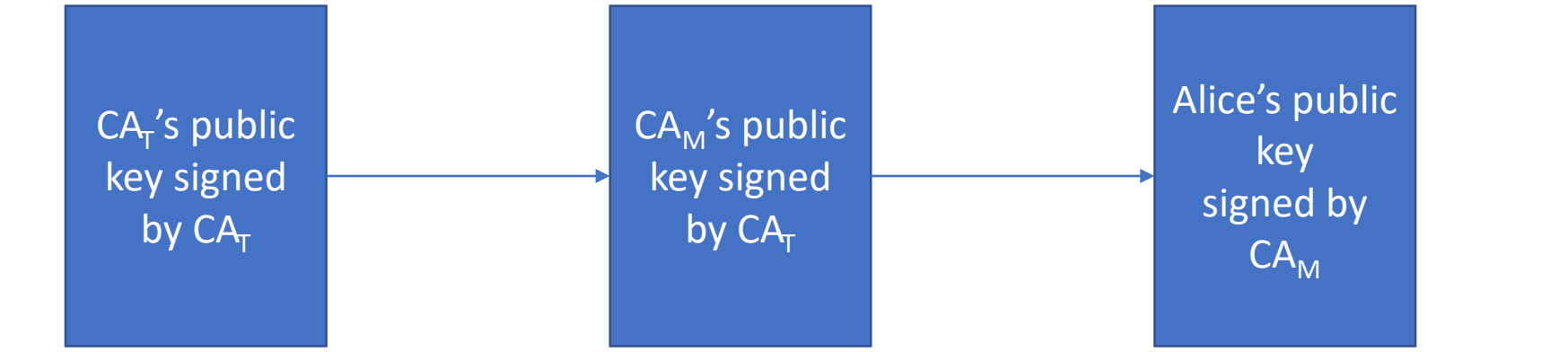
But Wait…

- How did Bob get Trent's public key?
  - We haven't solved anything… just moved the problem to Trent
- But… Trent can be a trusted third party for many users
- Consider:
  - IT department at VTC as "Trent"
  - Students authenticate to the VTC helpdesk to get a certificate
  - Students get a copy of VTC IT's public key at that time (*mutual authentication*)
  - Students/faculty can send encrypted messages to each other without prior arrangements
  - … as long as we trust VTC's IT department to sign keys appropriately

Certificate Authorities



Certificate Chains



Bob verifies certificate chain

Example

- Consider:
  - Alice is at UCLA. Bob is at VTC
  - Alice's key is certified by the UCLA IT department
  - The UCLA IT department's key is certified by Thawte (a commercial certificate authority)
  - Bob does not know Alice personally; downloads her key from her web page
  - Bob has the Thawte public key (it's common knowledge)
  - Bob trusts Thawte
  - Bob trusts the UCLA IT department
  - Bob believes he has the correct key for Alice

Public Key Algorithms…

- RSA
  - Depends on the difficulty of factoring very large prime numbers
- ElGamal
  - Depends on the difficulty of the “discrete logarithm problem”
- ECC (Elliptic Curve Cryptography)
  - Depends on the difficulty of the “elliptic curve discrete logarithm problem.” ECC is much more efficient than either RSA or ElGamal for a given level of security.
- DSA (Digital Signature Algorithm)
  - A standard approach for creating signatures using the ElGamal system

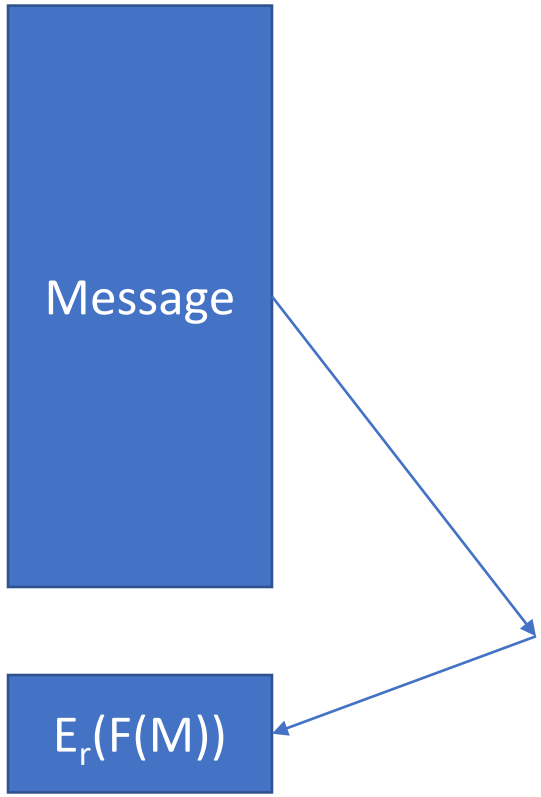
Cryptographic Hash Functions

- A Cryptographic hash function (or just “hash function”)…
  - … is like a super checksum
  - Mallory can’t not feasibly modify the message so that it produces the same *hash value*.
- Let
  - M be the message; F by the hash function; H be the hash value
  - Then we can write: *H = F(M)*
- Typical hash values are large (like 160 bits or more)
  - … but still small compared to the message being hashed (usually)

Hash Function Behavior

- Pre-Image Resistance
  - Given H, it should be infeasible to find an M such that H = F(M)
- Collision Resistance
  - Given an H = F(M) for a given M, it should be infeasible to find a second message M_2 such that H = F(M_2)
  - Note that many such messages exist; but it should be difficult to find any
  - Mallory should be unable to modify M to some M_2 with the same H
- Strong Collision Resistance
  - It should be infeasible to find two messages M_1 and M_2 such that H = F(M_1) and H = F(M_2)

Digital Signatures



Signer computes H = F(M); Encrypts H with private key

Verifier computes H = F(M); Decrypts signature with public key

It’s a Good System

- Features
  - Mallory can’t change message to have the same hash; any modifications are detected because the signature won’t verify
  - The signature won’t work on another document; Mallory can’t find a document with the same hash
  - Decrypting with the signer’s public key authenticates the document’s origin
    - Assuming you have the right public key, of course
  - Computing hash functions is quick. Only a small amount of data is processed with the slow public key algorithm
- *Document is authenticated and verified correct!*

64 Bit Hashes Too Small

- Any reasonable hash function generates 128 bit hashes or more
- Today mostly people use 160 bit or 256 bit hashes
- Consider what a 160 bit hash means for Alice
  - She has to do 2^{80} hash calculations to make a table with 2^{80} hash values, each 20 bytes in size (so 20 * 2^{80} bytes)
  - She has to do another 2^{80} hash calculations and look up each has value in the previously made table
  - *Very difficult!*
- Notice: Birthday Attack independent of hash function
  - Amounts to “brute force” for the hash function world

Hash Functions…

- MD5, SHA-1
  - Older and no longer considered secure
- SHA-2
  - Umbrella name for a family of related hash functions: SHA-224, SHA-256, SHA-384, and SHA-512
- SHA-3
  - Built on the Keccak family of “sponge” algorithms

Random Bits

- Our definition of a random bit sequence:
  - *No amount of knowledge of the previous bits in the sequence allows you to predict the next bit in the sequence with probability greater than 0.5*.
  - Emphasis on unpredictability; *everything else follows from this property*
  - You can produce random integers† from random bits by "filling up" the bits of the integer from the random bit generator.
  - You can produce random bits from random integers by reading the bits out of each random integer

† Here I'm talking about integers composed of a fixed range of bits (8 bits, 16 bits, etc)

Statistical Analysis

- Detecting randomness requires statistical arguments
  - In a large random sequence there should be “about” as many zeros as ones
    - But probably not exactly the same number…
    - … if given 1,000,000 bits it would be suspicious if there were exactly 500,000 zero bits
    - Can only express a confidence level in the hypothesis
  - 0101010101010101010101010101010101010101010101010101010101
    - The above has equal number of zeros and ones, but is unlikely to be random (though not impossible)
    - Consider frequency analysis… random sequences are “white noise” with equal “power” at all frequencies
  - Many other statistical tests are possible. Random sequences **pass them all!**

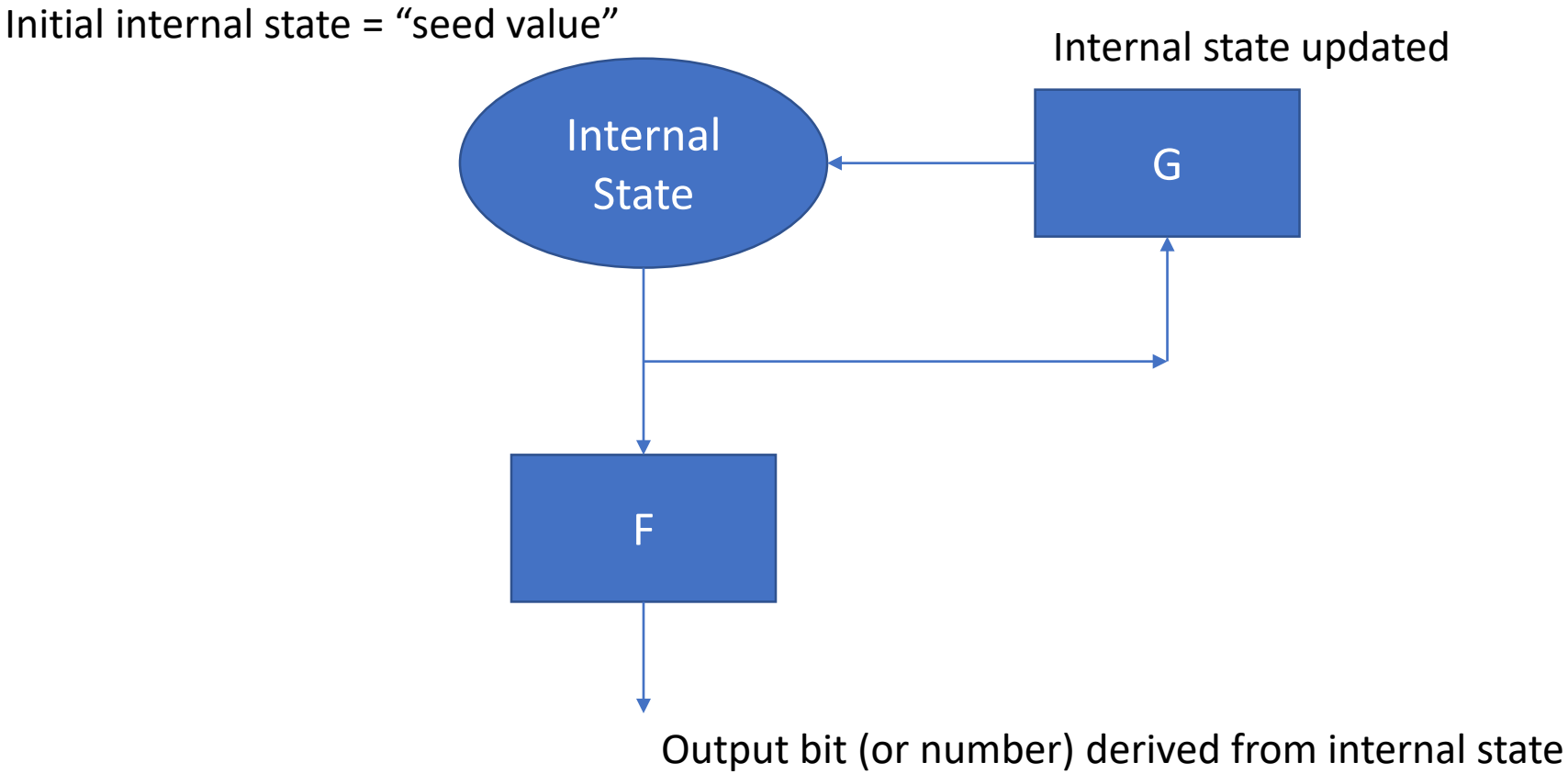
Two Kinds of Generators

- True Random Number Generators (TRNGs or just RNGs)
  - Produce bits using some underlying physical process
  - Ultimately relies on the fundamental randomness of quantum mechanics
    - Electrical noise
    - Radioactive decay
    - Turbulent flow
- Pseudo Random Number Generators (PRNGs)
  - Produce bits using a seed value and a deterministic algorithm
  - Technically not random at all: completely predictable
    - If you know the seed, you can calculate the entire sequence

Pros and Cons

- TRNGs
  - Require some kind of physical apparatus
  - Tend to produce random bits slowly
  - Still requires care to remove biases
    - No circuit can respond to all frequencies, for example, so some post processing necessary when using electrical noise to compensate for that.
- PRNGs
  - Can be implemented in software without physical hardware
  - Tend to produce random bits quickly
  - Requires a good algorithm to produce “statistically convincing” randomness

PRNG Block Diagram



Typical Security Application

- Alice wishes to encrypt a file for Bob
  - *Alice generates a "random" session key* for a symmetric algorithm such as AES
  - *Alice encrypts the session key* with Bob's public key
  - Alice encrypts the file with the session key
  - Alice sends (encrypted) file and (encrypted) session key to Bob
  - *Bob decrypts the session key* with his private key
  - Bob decrypts the file using the session key and AES

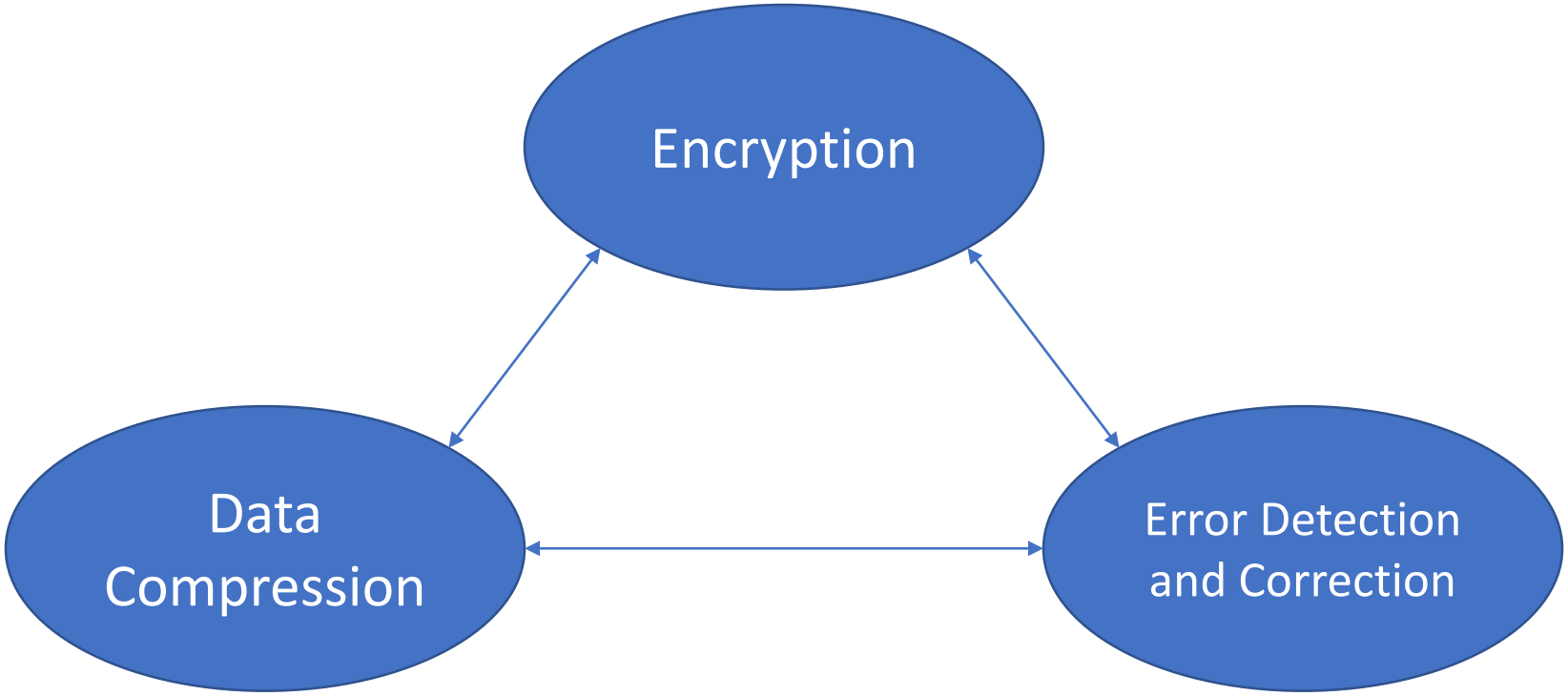
Enter: Mallory

- Mallory can't break the public key or symmetric key algorithms
- Mallory attacks the random number generator…
  - Mallory gets several legitimate files from Alice
  - Mallory takes note of the session keys
    - *The session keys are part of the random sequence produced by Alice's RNG*
  - Mallory tries to predict the next bits of Alice's RNG
  - Mallory intercepts (encrypted) message to Bob
  - Mallory uses guess of session key to decrypt the file
- Mallory doesn't have to get it right the first time
  - *If she can guess the session key in, say, 2^{64} tries that is not too hard!*

Pseudo-Random Number Generators…

- CryptoMT
  - The cryptographic version of the Mersenne Twister. (The normal MT, although very good statistically, is predictable after observing just 624 output values)
- Blum Blum Shub
  - Strength depends on the difficulty of factoring large primes
- Block Cipher in Counter Mode
  - Using a truly random key
- Stream Cipher
  - Stream ciphers work by generating a stream of pseudo random numbers that are XORed against the plaintext

The Three Gnomes



The Three Gnomes

- Encryption…
  - Breaks up patterns and structure in the input
  - Produces output that resembles random data (very high information content)
- Data Compression…
  - Attempts to use patterns and structure to find more compact representation
  - Produces output that resembles random data (very high information content)
- Error Detection and Correction…
  - Adds patterns and structure to help detect and correct errors
  - Produces redundant and non-random output

Rules

- When using multiple gnomes together…
  - Always compress before encrypting
    - Encrypted data is totally uncompressible
    - Compressed plain text is harder to crack… the encryption is hardened
  - Encrypt before applying error detection/correction
    - Encrypting error corrected data may weaken the encryption
    - … but applying error correction to cipher text means the plain text won't have it
  - Remember: *Encryption does not provide data integrity protection!*
  - Error corrected data is more compressible
  - ... but compressed data is more fragile
  - Encrypted data is more fragile