Lecture 14, April 29, 2026

Certificates

- Create token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key e_{Alice}
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)

signed by trusted authority (here, Cathy)

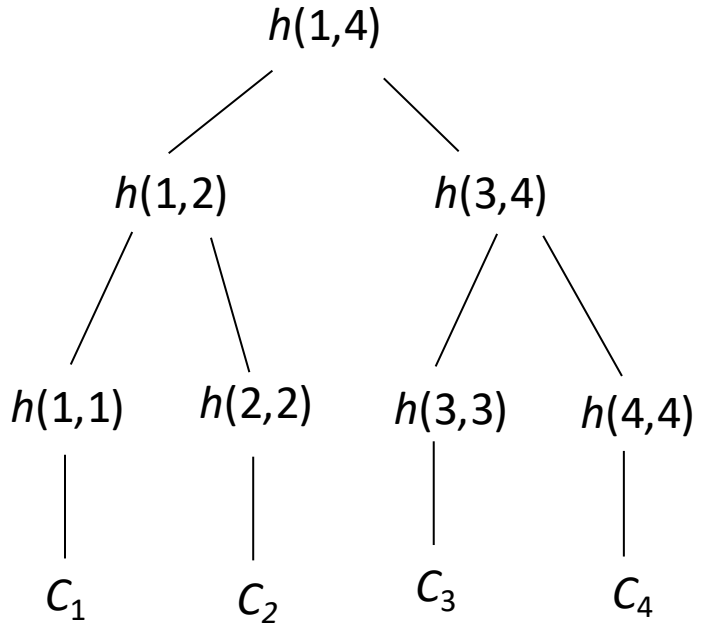
$$C_{\text{Alice}} = \{ e_{\text{Alice}} \,||\, \text{Alice} \,||\, T \}\, d_{\text{Cathy}}$$

Use

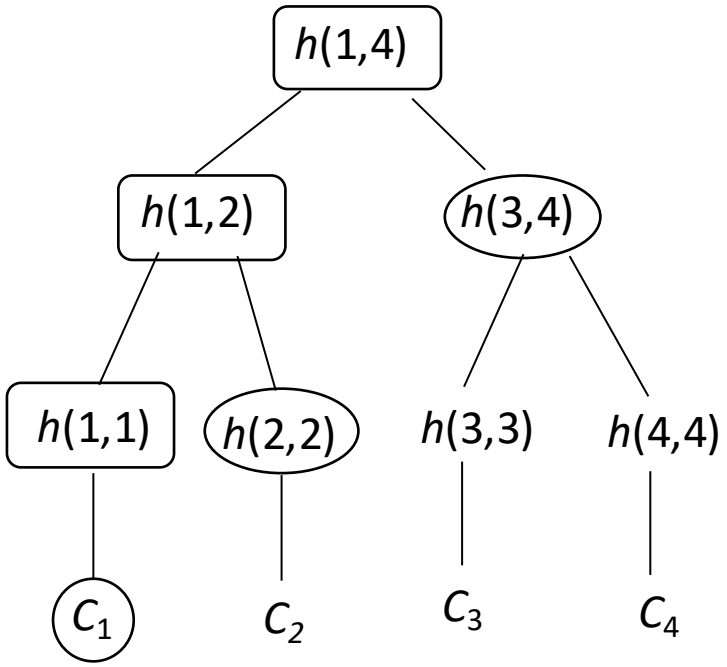
- Bob gets Alice's certificate
  - If he knows Cathy's public key, he can decipher the certificate
    - When was certificate issued?
    - Is the principal Alice?
  - Now Bob has Alice's public key
- Problem: Bob needs Cathy's public key to validate certificate
  - Problem pushed "up" a level
  - Two approaches: Merkle's tree, signature chains

Merkle's Tree Scheme

- Keep certificates in a file
  - Changing any certificate changes the file
  - Use crypto hash functions to detect this
- Define hashes recursively
  - h is hash function
  - C_i is certificate i
- Hash of file ($h(1,4)$ in example) known to all



Validation



- To validate C_1:
  - Compute $h(1, 1)$
  - Obtain $h(2, 2)$
  - Compute $h(1, 2)$
  - Obtain $h(3, 4)$
  - Compute $h(1,4)$
  - Compare to known $h(1, 4)$
- Need to know hashes of children of nodes on path that are not computed
- In drawing at left:
  - Circle contains what is to be validated
  - Ovals are what are to be obtained
  - Curved rectangles are what are to be computed

Problem

- File must be available for validation
  - Otherwise, can’t recompute hash at root of tree
  - Intermediate hashes would do
- Not practical in most circumstances
  - If any public key changed, validation fails unless tree is updated
  - This includes compromised certificates as well as legitimate public key changes
  - If copies of tree are widely distributed, a change to one must be reflected by all

Certificate Signature Chains

- Create certificate
  - Generate hash of certificate
  - Encipher hash with issuer's private key
- Validate
  - Obtain issuer's public key
  - Decipher enciphered hash
  - Recompute hash from certificate and compare
- Problem: getting issuer's public key

X.509 Chains

- Some certificate components in X.509v3:
  - Version
  - Serial number
  - Signature algorithm identifier: hash algorithm
  - Issuer's name; uniquely identifies issuer
  - Interval of validity
  - Subject's name; uniquely identifies subject
  - Subject's public key
  - Signature: enciphered hash

X.509 Certificate Validation

- Obtain issuer's public key
  - The one for the particular signature algorithm
- Decipher signature
  - Gives hash of certificate
- Recompute hash from certificate and compare
  - If they differ, there's a problem
- Check interval of validity
  - This confirms that certificate is current

Issuers

- *Certification Authority* (*CA*): entity that issues certificates
  - Multiple issuers pose validation problem
  - Alice's CA is Cathy; Bob's CA is Dan; how can Alice validate Bob's certificate?
  - Have Cathy and Dan cross-certify by issuing certificates for each other

Validation and Cross-Certifying

- Notation: *X*<<*Y*>> means *X* issues certificate for *Y*
- Certificates:
  - Cathy<<Alice>>
  - Dan<<Bob>
  - Cathy<<Dan>>
  - Dan<<Cathy>>
- Alice validates Bob's certificate
  - Alice obtains Cathy<<Dan>>
  - Alice uses (known) public key of Cathy to validate Cathy<<Dan>>
  - Alice uses Cathy<<Dan>> to validate Dan<<Bob>>

PGP Chains

- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (present in version 3 only)
  - Public key algorithm, associated parameters
  - Public key

OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer's key identifier (identifies key to encipher hash)
  - Public key algorithm (used to encipher hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (enciphered hash)
- Version 4 packet more complex

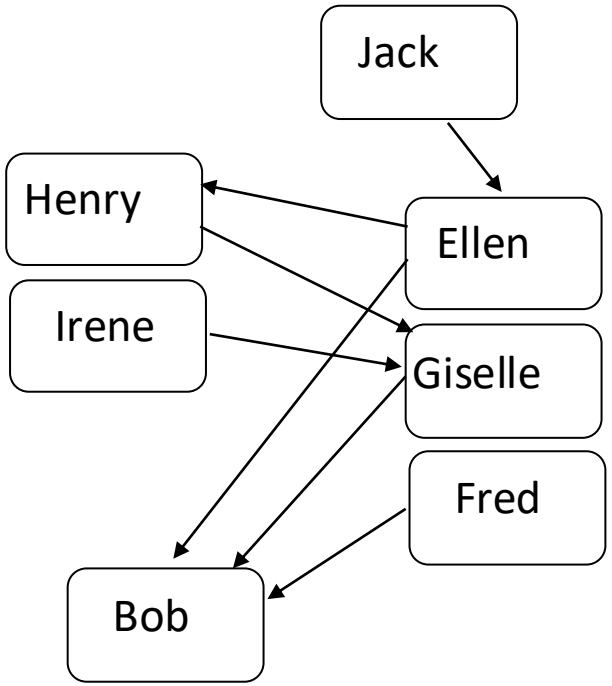
Signing

- Single certificate may have multiple signatures
- Notion of “trust” embedded in each signature
  - Range from “untrusted” to “ultimate trust”
  - Signer defines meaning of trust level (no standards!)
- All version 4 keys signed by the subject of the certificate
  - Called “self-signing”
  - Version 3 certificates can be too

Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Does not know Fred, Giselle, or Ellen
- Alice gets Giselle's cert
  - Knows Henry slightly, but his signature is at "casual" level of trust
- Alice gets Ellen's cert
  - Knows Jack, so uses his cert to validate Ellen's, then hers to validate Bob's

Arrows show signatures
Self signatures not shown



Public Key Infrastructures (PKIs)

- An infrastructure that manages public keys and certificate authorities
  - This includes registration authorities and other entities involved in creating and issuing certificates

PKI Problems

Basis for any PKI is trust

- Trust that the binding of identity to public key is correct
  - Degree of confidence depends on CA or RA
- Trust that appropriate CA issued the certificate
  - Also that issuance policies are understood
  - Also that implementation of signing, and PKI mechanisms,
- Certificate does not embody authorization
  - Identity may, but that is external to PKI
- Trust that no 2 certificates will have same public (and hence private) key

Key Revocation

- Certificates invalidated *before* expiration
  - Usually due to compromised key
  - May be due to change in circumstance (*e.g.,* someone leaving company)
- Problems
  - Entity revoking certificate authorized to do so
  - Revocation information circulates to everyone fast enough
    - Network delays, infrastructure problems may delay information

CRLs

- *Certificate revocation list* lists certificates that are revoked
- X.509: only certificate issuer can revoke certificate
  - Added to CRL
- PGP: signers can revoke signatures; owners can revoke certificates, or allow others to do so
  - Revocation message placed in PGP packet and signed
  - Flag marks it as revocation message

Problems Using Cryptography

- Using cipher requires knowledge of environment, and threats in the environment, in which cipher will be used
  - Is the set of possible messages small?
  - Can an active wiretapper rearrange or change parts of the message?
  - Do the messages exhibit regularities that remain after encipherment?
  - Can the components of the message be misinterpreted?

Attack #1: Precomputation

- Set of possible messages *M* small
- Public key cipher *f* used
- Idea: precompute set of possible ciphertexts *f*(*M*), build table (*m*, *f*(*m*))
- When ciphertext *f*(*m*) appears, use table to find *m*
- Also called *forward searches*

Example

- Cathy knows Alice will send Bob one of two messages: enciphered BUY, or enciphered SELL
- Using public key e_{Bob}, Cathy precomputes

  $m_1 = \{ \text{BUY} \}\, e_{Bob}$, $m_2 = \{ \text{SELL} \}\, e_{Bob}$
- Cathy sees Alice send Bob m_2
- Cathy knows Alice sent SELL

Misordered Blocks

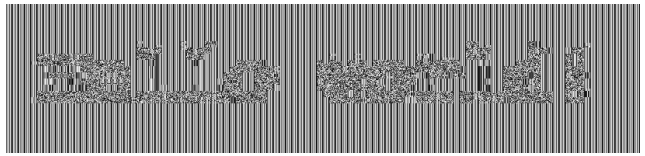
- Alice sends Bob message
  - n_{Bob} = 262631, e_{Bob} = 45539, d_{Bob} = 235457
- Message is TOMNOTANN (191412 131419 001313)
- Enciphered message is 193459 029062 081227
- Eve intercepts it, rearranges blocks
  - Now enciphered message is 081227 029062 193459
- Bob gets enciphered message, deciphers it
  - He sees ANNNOTTOM, opposite of what Alice sent

Solution

- Digitally signing each block won't stop this attack
- Two approaches:
  - Cryptographically hash the *entire* message and sign it
  - Place sequence numbers in each block of message, so recipient can tell intended order; then sign each block

Statistical Regularities

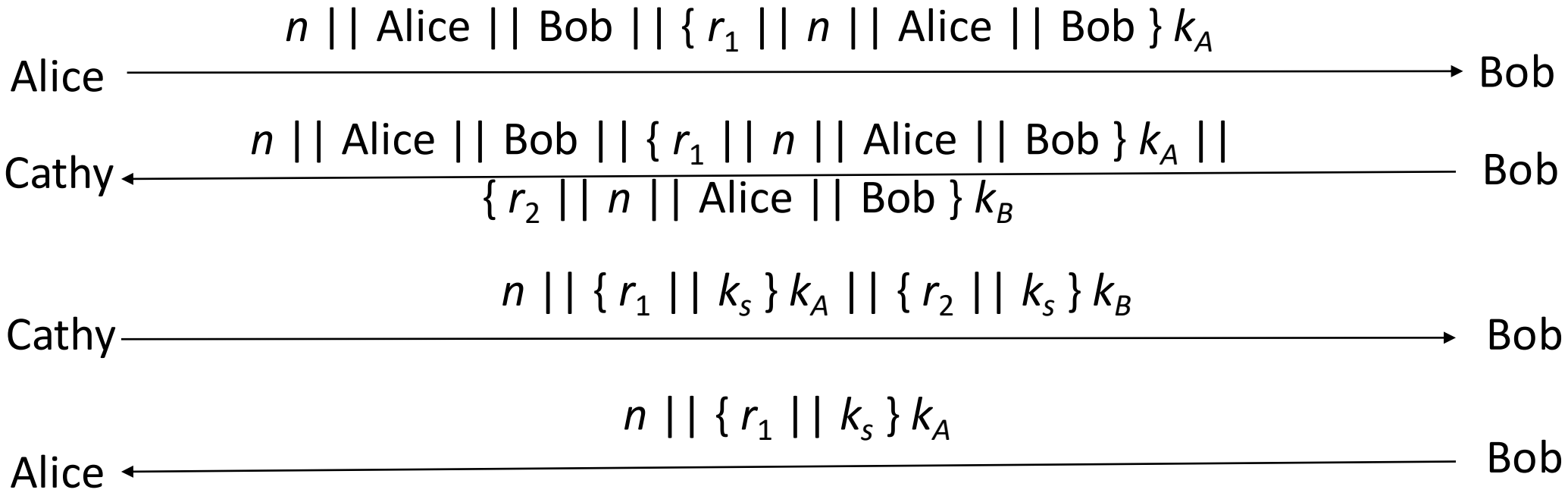
- If plaintext repeats, ciphertext may too
- Example using AES-128:
  - Input image: Hello world!
  - corresponding output image:



  - Note you can still make out the words
- Fix: cascade blocks together (chaining) More details later

Type Flaw Attacks

- Assume components of messages in protocol have particular meaning
- Example: Otway-Rees:



The Attack

- Ichabod intercepts message from Bob to Cathy in step 2
- Ichabod *replays* this message, sending it to Bob
  - Slight modification: he deletes the cleartext names
- Bob *expects* $n \,||\, \{ r_1 \,||\, k_s \} k_A \,||\, \{ r_2 \,||\, k_s \} k_B$
- Bob *gets* $n \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_A \,||\, \{ r_2 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \} k_B$
- So Bob sees $n \,||\, \text{Alice} \,||\, \text{Bob}$ as the session key — and Ichabod knows this
- When Alice gets her part, she makes the same assumption
- Now Ichabod can read their encrypted traffic

Solution

- Tag components of cryptographic messages with information about what the component is
  - But the tags themselves may be confused with data …

What These Mean

- Use of strong cryptosystems, well-chosen (or random) keys not enough to be secure
- Other factors:
  - Protocols directing use of cryptosystems
  - Ancillary information added by protocols
  - Implementation (not discussed here)
  - Maintenance and operation (not discussed here)