General description

- Public-key encryption (generalizes block cipher)
  - Separate encryption key $E_K$ (public) and decryption key $D_K$ (secret)
- Signature scheme (generalizes MAC)
  - Separate signing key $S_K$ (secret) and verification key $V_K$ (public)
RSA setup
- Choose $n = pq$, product of two large primes, as modulus
- $n$ is public, but $p$ and $q$ are secret
- Compute encryption and decryption exponents $e$ and $d$ such that
  $$M^{ed} = M \pmod{n}$$

RSA encryption
- Public key is $(n, e)$
- Encryption of $M$ is $C = M^e \pmod{n}$
- Secret key is $(n, d)$
- Decryption of $C$ is $C^d = M^{ed} = M \pmod{n}$

RSA signature
- Signing key is $(n, d)$
- Signature of $M$ is $S = M^d \pmod{n}$
- Verification key is $(n, e)$
- Check signature by $S^e = M^{de} = M \pmod{n}$
- Note: symmetry is a nice feature of RSA, not shared by other systems

RSA and factoring
- We’re not sure factoring is hard (likely not even NP-complete), but it’s been unsolved for a long time
- If factoring is easy (e.g., in P), RSA is insecure
- Converse might not be true: RSA might have other problems

Aside: stronger reduction
- Public-key algorithms actually equivalent to factoring and discrete log exist
- But not widely used because of speed or other efficiency issues
- Even symmetric-key algorithms with such security
  - But they’re much less efficient than AES et al.
**Homomorphism**
- Multiply RSA ciphertexts $\Rightarrow$ multiply plaintexts
- This *homomorphism* is useful for some interesting applications
- Even more powerful: fully homomorphic encryption (e.g., both $+$ and $\times$)
  - First demonstrated in 2009; still very inefficient

**Problems with vanilla RSA**
- Homomorphism leads to chosen-ciphertext attacks
- If message and $e$ are both small compared to $n$, can compute $M^{1/e}$ over the integers
- Many more complex attacks too

**Hybrid encryption**
- Public-key operations are slow
- In practice, use them just to set up symmetric session keys
  - Only pay RSA costs at setup time
  - Breaks at either level are fatal

**Padding, try #1**
- Need to expand message (e.g., AES key) size to match modulus
- PKCS#1 v. 1.5 scheme: prepend 00 01 FF FF .. FF
- Surprising discovery (Bleichenbacher'98): allows adaptive chosen ciphertext attacks on SSL

**Modern “padding”**
- Much more complicated encoding schemes using hashing, random salts, Feistel-like structures, etc.
- Common examples: OAEP for encryption, PSS for signing
- Progress driven largely by improvement in random oracle proofs

**Simpler padding alternative**
- "Key encapsulation mechanism" (KEM)
- For common case of public-key crypto used for symmetric-key setup
  - Also applies to DH
- Choose RSA message $r$ at random $\mod n$, symmetric key is $H(r)$
  - Hard to retrofit, RSA-KEM insecure if $e$ and $r$ reused with different $n$
Box and locks revisited

- Alice and Bob's box scheme fails if an intermediary can set up two sets of boxes
- Real world analogue: challenges of protocol design and public key distribution

Outline

Public key encryption and signatures
Announcements
Cryptographic protocols
More causes of crypto failure

Upcoming assignments

- HA2: can start registering groups
  - Send email to TA
  - Tell us even if same group as HA1
- Project progress report: due Wednesday 11/5
- Exercise set 3: due Thursday 11/6

A couple more security goals

- Non-repudiation: principal cannot later deny having made a commitment
  - I.e., consider proving fact to a third party
- Forward secrecy: recovering later information does not reveal past information
  - Motivates using Diffie-Hellman to generate fresh keys for each session

Abstract protocols

- Outline of what information is communicated in messages
  - Omit most details of encoding, naming, sizes, choice of ciphers, etc.
- Describes honest operation
  - But must be secure against adversarial participants
- Seemingly simple, but many subtle problems
Protocol notation

\[ A \rightarrow B : N_B, \{T_0, B, N_B\}_{K_B} \]
- **A \rightarrow B**: message sent from Alice intended for Bob
- B (after \(\rightarrow\)): Bob's name
- \(\{\cdots\}_K\): encryption with key \(K\)

Example: simple authentication

\[ A \rightarrow B : A, \{A, N\}_{K_A} \]
- E.g., Alice is key fob, Bob is garage door
- Alice proves she possesses the pre-shared key \(K_A\)
- Without revealing it directly
- Using encryption for authenticity and binding, not secrecy

Nonce

\[ A \rightarrow B : A, \{A, N\}_{K_A} \]
- \(N\) is a nonce: a value chosen to make a message unique
- Best practice: pseudorandom
- In constrained systems, might be a counter or device-unique serial number

Replay attacks

- A nonce is needed to prevent a verbatim replay of a previous message
- Garage door difficulty: remembering previous nonces
- Particularly: lunchtime/roommate/valet scenario
- Or, door chooses the nonce: challenge-response authentication

Man-in-the-middle attacks

- Gender neutral: middleperson attack
- Adversary impersonates Alice to Bob and vice-versa, relays messages
- Powerful position for both eavesdropping and modification
- No easy fix if Alice and Bob aren't already related

Chess grandmaster problem

- Variant or dual of MITM
- Adversary forwards messages to simulate capabilities with his own identity
- How to win at correspondence chess
- Anderson's MiG-in-the-middle
Needham-Schroeder

Authenticated key exchange assuming public keys (core):

\[ A \rightarrow B : \{N_A, A\}_K_B \]
\[ B \rightarrow A : \{N_A, N_B\}_K_A \]
\[ A \rightarrow B : \{N_B\}_K_B \]

Needham-Schroeder MITM

\[ A \rightarrow C : \{N_A, A\}_K_C \]
\[ C \rightarrow B : \{N_A, A\}_K_B \]
\[ B \rightarrow C : \{N_A, N_B\}_K_A \]
\[ C \rightarrow A : \{N_A, N_B\}_K_A \]
\[ A \rightarrow C : \{N_B\}_K_C \]
\[ C \rightarrow B : \{N_B\}_K_B \]

Certificates, Denning-Sacco

A certificate signed by a trusted third-party \( S \) binds an identity to a public key:

\[ C_A = \text{Sign}_S(A, K_A) \]

Suppose we want to use \( S \) in establishing a session key \( K_{AB} \):

\[ A \rightarrow S : A, B \]
\[ S \rightarrow A : C_A, C_B \]
\[ A \rightarrow B : C_A, C_B, \{\text{Sign}_A(K_{AB})\}_K_B \]

Attack against Denning-Sacco

\[ A \rightarrow S : A, B \]
\[ S \rightarrow A : C_A, C_B \]
\[ A \rightarrow B : C_A, C_B, \{\text{Sign}_A(K_{AB})\}_K_B \]
\[ B \rightarrow S : B, C \]
\[ S \rightarrow B : C_B, C_C \]
\[ B \rightarrow C : C_A, C_C, \{\text{Sign}_A(K_{AB})\}_K_C \]

By re-encrypting the signed key, Bob can pretend to be Alice to Charlie

Envelopes analogy

Encrypt then sign, or vice-versa?

On paper, we usually sign inside an envelope, not outside. Two reasons:

- Attacker gets letter, puts in his own envelope (c.f. attack against X.509)
- Signer claims “didn’t know what was in the envelope” (failure of non-repudiation)

Design robustness principles

- Use timestamps or nonces for freshness
- Be explicit about the context
- Don’t trust the secrecy of others’ secrets
- Whenever you sign or decrypt, beware of being an oracle
- Distinguish runs of a protocol
Implementation principles

- Ensure unique message types and parsing
- Design for ciphers and key sizes to change
- Limit information in outbound error messages
- Be careful with out-of-order messages

Outline

- Public key encryption and signatures
- Announcements
- Cryptographic protocols
- More causes of crypto failure

Random numbers and entropy

- Cryptographic RNGs use cipher-like techniques to provide indistinguishability
- But rely on truly random seeding to stop brute force
  - Extreme case: no entropy → always same “randomness”
- Modern best practice: seed pool with 256 bits of entropy
  - Suitable for security levels up to $2^{256}$

Netscape RNG failure

- Early versions of Netscape SSL (1994-1995) seeded with:
  - Time of day
  - Process ID
  - Parent process ID
- Best case entropy only 64 bits
  - (Not out of step with using 40-bit encryption)
- But worse because many bits guessable

Debian/OpenSSL RNG failure (1)

- OpenSSL has pretty good scheme using /dev/urandom
- Also mixed in some uninitialized variable values
  - “Extra variation can’t hurt”
- From modern perspective, this was the original sin
  - Remember undefined behavior discussion?
- But had no immediate ill effects

Debian/OpenSSL RNG failure (2)

- Debian maintainer commented out some lines to fix a Valgrind warning
  - “Potential use of uninitialized value”
- Accidentally disabled most entropy (all but 16 bits)
- Brief mailing list discussion didn’t lead to understanding
- Broken library used for ~2 years before discovery
Detected RSA/DSA collisions
- Up to about 1% of the SSL and SSH keys on the public net are breakable
  - Some sites share complete keypairs
  - RSA keys with one prime in common (detected by large-scale GCD)
- One likely culprit: insufficient entropy in key generation
  - Embedded devices, Linux /dev/urandom vs. /dev/random
- DSA signature algorithm also very vulnerable

Side-channel attacks
- Timing analysis:
  - Number of 1 bits in modular exponentiation
  - Unpadding, MAC checking, error handling
  - Probe cache state of AES table entries
- Power analysis
  - Especially useful against smartcards
- Fault injection
- Data non-erasure
  - Hard disks, “cold boot” on RAM

WEP “privacy”
- First WiFi encryption standard: Wired Equivalent Privacy (WEP)
- F&S: designed by a committee that contained no cryptographers
- Problem 1: note “privacy”: what about integrity?
  - Nope: stream cipher + CRC = easy bit flipping

WEP shared key
- Single key known by all parties on network
- Easy to compromise
- Hard to change
- Also often disabled by default
- Example: a previous employer

WEP key size and IV size
- Original sizes: 40-bit shared key (export restrictions) plus 24-bit IV = 64-bit RC4 key
  - Both too small
- 128-bit upgrade kept 24-bit IV
  - Vague about how to choose IVs
  - Least bad: sequential, collision takes hours
  - Worse: random or everyone starts at zero

WEP RC4 related key attacks
- Only true crypto weakness
- RC4 “key schedule” vulnerable when:
  - RC4 keys very similar (e.g., same key, similar IV)
  - First stream bytes used
- Not a practical problem for other RC4 users like SSL
  - Key from a hash, skip first output bytes
Trustworthiness of primitives

- Classic worry: DES S-boxes
- Obviously in trouble if cipher chosen by your adversary
- In a public spec, most worrying are unexplained elements
- Best practice: choose constants from well-known math, like digits of \( \pi \)

Dual_EC_DRBG (1)

- Pseudorandom generator in NIST standard, based on elliptic curve
- Looks like provable (slow enough!) but strangely no proof
- Specification includes long unexplained constants
- Academic researchers find:
  - Some EC parts look good
  - But outputs are statistically distinguishable

Dual_EC_DRBG (2)

- Found 2007: special choice of constants allows prediction attacks
  - Big red flag for paranoid academics
- Significant adoption in products sold to US govt. FIPS-140 standards
  - Semi-plausible rationale from RSA (EMC)
- NSA scenario basically confirmed recently by Snowden leaks
  - NIST and RSA immediately recommend withdrawal

Next time

- Crypto in SSH, TLS, DNSSEC
- Public-key infrastructure