Week 5

Breaking RSA, Digital Signatures and Authentication

THE UNIVERSITY OF SYDNEY

Review: RSA

- RSA is an asymmetric encryption algorithm with authentication. Key generation is done by a single party, Alice:
  - 1. Generate two large primes p, q, and define modulus $n = pq$.
  - 2. Select some e coprime to $\phi(n) = (p-1)(q-1)$.
  - 3. Find d such that $e \times d \equiv 1 \pmod{\phi(n)}$.
  - 4. The public key is (n, e) and the private key is (n, d).
  - 5. p, q, $\phi(n)$ are no longer needed.
- If Bob now wants to send the message $m \in Z_n^\times$ to Alice:
  - 1. Bob computes a ciphertext $C = m^e \pmod{n}$ and sends the result to Alice.
  - 2. Alice receives m^e and decrypts the ciphertext to recover the message by computing $(m^e)^d \equiv m^{ed} \equiv m \pmod{n}$. Recall that we found d such that $e \times d \equiv 1 \pmod{\phi(n)}$.

Chinese Remainder Theorem

- The Chinese Remainder Theorem (CRT) can be used to attack the RSA public key (n, e) by recording ciphertext messages with the same e value and different n values.

- If Alice sends the same ciphertext message, m, to three recipients with different public keys (n_1, e), (n_2, e), (n_3, e), Eve can solve the following system of simultaneous equations using CRT to find m^e:

$$C_1 \equiv m^e \bmod n_1$$
$$C_2 \equiv m^e \bmod n_2$$
$$C_3 \equiv m^e \bmod n_3$$

  where $\gcd(n_i, n_j) = 1$ (i.e. n_i and n_j are relatively prime).
- Since e is known, we can use logarithms to recover m from m^e

Small Encryption Exponent Attack

- The small encryption exponent attack, or Coppersmith's attack, exploits a bad choice of encryption exponent which results in $m^e < n$.
- If $m^e < n$:

$$C = m^e \ (mod\ n) = m^e$$

- The message can be calculated by taking the integer e^{th} root of C

- A small encryption exponent is often chosen in order to improve the speed of RSA encryption.

- A small encryption exponent should not be used to send the same message (or the same message with variation) to several entities. Salting the plaintext (padding with random bits) can help avoid this attack.

Other Attacks on RSA

- Small Decryption Exponent
  - Common in small devices, since a small d makes for efficient decryption.
  - A small decryption exponent should also be avoided since there are algorithms to compute d from (n, e) if d is less than approximately one-quarter as many bits as the modulus, n.

- Forward Search Attack
  - Since the encryption key is public, if the message space is small or predictable, an attacker can try to brute force on the message space.
  - Salting the plaintext by adding random numbers may help prevent this attack
  - Example: A stock trading system
  - Message format: "{BUY, SELL} {DDDD} {TICKER}" (only 1000 tickers)
    - $|M| = 2 \times 10000 \times 1000 = 20,000,000$ possible messages
    - At 1 million guesses / second $\rightarrow$ 20 seconds to guess transmitted message.

Homomorphic properties of RSA

- RSA encryption is homomorphic. Suppose:

$$c_1 = m_1^e \bmod n$$
$$c_2 = m_2^e \bmod n$$

- Then:

$$c_1 \times c_2 = (m_1 \times m_2)^e \; mod\, n$$

- Using this property, we can attack RSA using an adaptive chosen ciphertext attack.

Homomorphic properties of RSA

- Suppose Eve wants Alice to reveal the decryption of a ciphertext c intended for another recipient, Bob:

$$c = m^e \, mod \, n$$

- Eve sends Alice an adapted ciphertext c':

$$(c') = cx^e \, mod \, n$$

where $x \in \mathrm{Z}_\mathrm{n}^\times$ is a randomly chosen “blinding factor”.

- Alice computes:

$$\begin{aligned} (c')^d &= (cx^e)^d \bmod n \\ &= c^d x^{ed} \bmod n \\ &= m x^{ed} \bmod n \\ &= mx \bmod n \end{aligned}$$

- If Alice reveals this information, Eve can “unblind” the message by computing:

$$(mx)x^{-1} \bmod n = m \bmod n$$

Digital Signatures

Signatures

- Signatures are used to bind an author to a document.
- Desirable properties for a signature:
  - **Authentic** Sufficient belief that the signer deliberately signed the document.
  - **Unforgeable** Proof that only the signer could have signed the document, no-one else.
  - **Non-reusable** The signature is intrinsically bound to the document and cannot be moved to another (i.e. be reused).
  - **Unalterable** The signature cannot be altered after signing.
  - **Non-repudiation** The signer cannot later deny that they did not sign it (important).
- As with all things, these properties can be attacked and subverted. We must consider such attacks when designing systems that use signatures.

Digital Signatures

- We have:

  m - The message to be signed

  k - The secret key

  F - The signature scheme (function)

  S - The signature

$$S = F(m,k)$$

- The message m is signed using the secret key k, known only to the signer, which binds the signature S to the message m using some signature scheme F.
- Given (m, S) anyone can verify the signature without the secret k.
- Non-repudiation is achieved through the secrecy of k.

Digital Signatures with Public Keys

- Say Alice wishes to sign a message and send it to Bob
- Generation of a key:
  - 1. Alice generates two keys: A_v public (verifying) A_s private (signing)
  - 2. A_v is published in a public directory
  - 3. A_s is kept secret
- Signature Generation:
  - 1. Alice chooses n random bits: $r = \{0,1\}^n$
  - 2. Alice hashes the message to get a message digest: $d = h(m)$
  - 3. Alice generates $S = \text{signature}(d, r, A_s)$
  - 4. Alice sends (m, S) to Bob
- Signature Verification:
  - 1. Bob obtains A_v from the public directory
  - 2. Bob computes $d = h(m)$
  - 3. Bob runs $\text{verify}(d, A_v, S)$

Preventing Signature Replay

- Why do we include $r = \{0,1\}^n$ in the signature?

- Consider the following scenario:
  - Alice sends Bob a digital cheque for $100.
  - Bob takes the cheque to the bank.
  - The bank verifies that the signature is valid and credits Bob's account.

- What is stopping Bob from cashing the same cheque twice? (i.e. perform a replay attack)
  - The random value r is known as a **nonce** and is used to avoid replay.
  - The bank keeps track of all nonces it has seen so far from Alice.

Signature Attack Models

- Universal Forgery
  - Attacker can recover private key A_s from public key A_v and (m, S)

- Selective Forgery
  - Attacker can forge signatures for a chosen message or class of message

- Existential Forgery
  - Attacker can forge a signature for a known message. Possible only in theory (based on currently available resources)

Signature based on RSA: Naïve protocol

- A naïve protocol based on RSA might be as follows.
- Key Generation:
  - $n = pq$ — p, q are large primes
  - $d \times e \equiv 1 \bmod \phi(n)$ — e coprime to $\phi(n)$
  - $A_v = (n, e)$ — public/verifying key
  - $A_s = (n, d)$ — private/signature key
- Signature Generation:
  - Assume $m \in \mathrm{Z}_{\mathrm{n}}^{\times}$
  - $S = m^d \; mod \; n$ — RSA decryption exponent
- Signature Verification:
  - $S^e = m \; mod \; n$ — RSA encryption exponent

Problem with Naïve protocol

- Eve can trick Alice into signing some message m.
- **Based on RSA's homomorphic property:**
  - If: $s_1 = m_1^d \pmod{n}$ and $s_2 = m_2^d \pmod{n}$
  - Then: $s_1 s_2 = (m_1 m_2)^d \bmod n$
- Attack on naïve RSA scheme:
  - 1. Eve wants Alice to sign a hidden message m
  - 2. Eve picks random $r \in \mathrm{Z}_\mathrm{n}^\times$
  - 3. Eve computes $m' = m \times r^e \pmod{n}$
  - 4. Eve asks Alice to sign m'
  - 5. Alice returns $s' = (m')^d \pmod{n} = m \times r^{ed} \pmod{n}$
  - 6. Eve computes $s = \frac{s'}{r} \pmod{n}$
- The pair (m, s) is a valid message signature pair! Eve tricked Alice into signing the hidden message m.

PKCS#1 Signature Scheme (RFC2313)

- Public Key Cryptography Standards #1
  - The verification function of the naïve RSA signature scheme processes the message.
  - PKCS#1 processes a hash instead (much faster)
- Signature Generation:
  - 1. $n = pq$ (1024-bit modulus)
  - 2. Alice calculates $D = h(m)$ (160-bit hash)
  - 3. Define encryption block:
    - EB = [00 | BT | PS | 00 | D]
      - PS: Padding to make the block the same length as the modulus
      - BT: Block type dictates padding style
      - EB is 864 bits + 160 bits = 1024 bits
  - 4. Alice calculates $S = \text{EB}^d \pmod{n}$
  - 5. Alice sends (S, m)

PKCS#1 Signature Scheme (RFC2313)

- Signature Verification:
  - $S \;=\; \mathrm{EB}^{d} \pmod{n}$
  - Bob calculates $S^{e} \bmod n \;=\; \mathrm{EB} \pmod{n}$
  - Bob checks the first 864 bits are valid
  - Bob checks the last 160 bits are valid (i.e. $=\; h(m)$)

Digital Signature Algorithm (DSA)

- The Digital Signature Algorithm (DSA) was selected by NIST in 1991 as the Digital Signature Standard (DSS). Parameter Selection:
  - Choose a hash function H
    - Originally SHA-1
    - Now SHA-2 is preferred
  - Choose key lengths N and L such that $N < L$
  - Choose an N-bit prime q
  - Choose an L-bit prime modulus p such that $p - 1$ is a multiple of q.
  - Choose a random integer h such that $1 < h < p - 1$
  - Compute $g = h^{p-1/q} \bmod p$. If $g = 1$, repeat with different h

Digital Signature Algorithm (DSA)

- Key Generation and Distribution:
  - Secret key x chosen randomly in $0 < x < q$
  - Compute public key: $y = g^x \pmod{p}$
    - Also provide p, q, and g parameters as part of public key.

- Signature Generation:
  - Pick random $k \in \mathrm{Z}_{\mathrm{n}}^{\times}$ where $1 < k < q$
    - Must be unique per message
  - Calculate $r = (g^k \bmod p) \bmod q$
    - If $r = 0$, repeat with different k
  - Calculate $s = \left(k^{-1}(H(m) + xr)\right) \bmod q$
    - If $s = 0$, repeat with different k
  - Signature is: (r, s)

Digital Signature Algorithm (DSA)

- Signature Verification:
  - Verify:
    - $0 < r < q$
    - $0 < s < q$
  - Calculate $w = s^{-1} \bmod q$
  - Calculate $u_1 = H(m) \times w \bmod q$
  - Calculate $u_2 = r \times w \bmod q$
  - Calculate $v = (g^{u_1} \times y^{u_2} \bmod p) \bmod q$
  - Signature is valid if $v == r$

- Correctness of DSA Digital Signature Algorithm - Wikipedia

Notes on DSA/DSS

- Security of DSA is based on the properties of random signature value k.
  - Repeating the same k, using a predictable k or leaking a few bits of k in each of several signatures, is enough to reveal the private key x.

- DSA signature verification can be sped up (by a factor of 2) by using simultaneous exponentiation.

- Signatures based on RSA are vulnerable to threat of quantum computing. Development of alternative digital signatures using one-way hash functions may be more quantum-resistant Lamport signature - Wikipedia

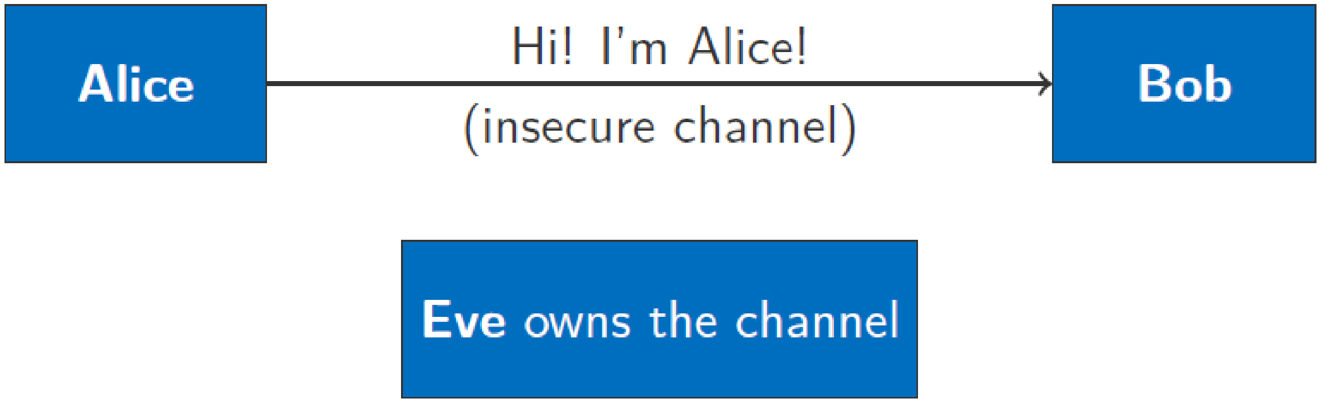
- Current standards forbid generating new signatures with DSA. However, verifying signatures generated with DSA is still accepted.

Authentication

THE UNIVERSITY OF SYDNEY

Authentication

– Core Problem: How does Bob know that Alice is not Eve?



Authentication

- Authentication is a means by which identity is established.
  - Bob needs to ensure that the party at the end of the channel is Alice.
    - Ensure Alice’s identity.
    - Ensure Alice has actively participated.
  - Goal: achieve this over an insecure channel with an active attacker, and no shared secrets.
  - Note: authentication must be combined with key exchange to avoid session hijacking (after authentication).

Objectives of Identification Protocols

- If Alice and Bob are both honest, Alice should be able to successfully authenticate herself, i.e. Bob will complete the protocol having verified Alice's identity.
  - Bob should not be able to reuse an identification exchange with Alice so as to impersonate her in conversations with others.
  - The probability that Eve can successfully impersonate Alice should be negligible (computationally hard).
  - All of the above should remain true if:
    - Eve has seen many previous authentication sessions between Alice and Bob
    - Eve has authenticated with either or both of Alice and Bob
    - Multiple authentication sessions are being run simultaneously

Basis of Identification

- Something you know:
  - Password, PIN, secret key, mother’s maiden name, colour of pet...
- Something you have:
  - Magnetic card, smart card, physical key, digital security key, a phone with Google Authenticator...
- Something you are:
  - Biometrics: DNA, signatures, fingerprints, voice, retinal patterns, hand geometry, typing dialect/profiling.
  - Biometrics have problems in real-world situations:
    - DNA and fingerprints are left everywhere.
    - How do you authenticate with a black eye\ bruised fingertip?

Examples of Authentication

- To verify identity as a precursor to communications:
  - Confirming the sender of the email before updating the card details on "amazon.com".
- To facilitate access to a resource:
  - Local/remote access to computing resources (password, OTP)
  - Withdrawal of money from an ATM (keycard and PIN)
  - Allow communications through a web server proxy
  - Allow physical access to restricted areas (swipe card)
  - Border crossing (passport, fingerprints)
- To facilitate resource tracking and billing:
  - Mobile phone access

Preventing Attacks on Authentication

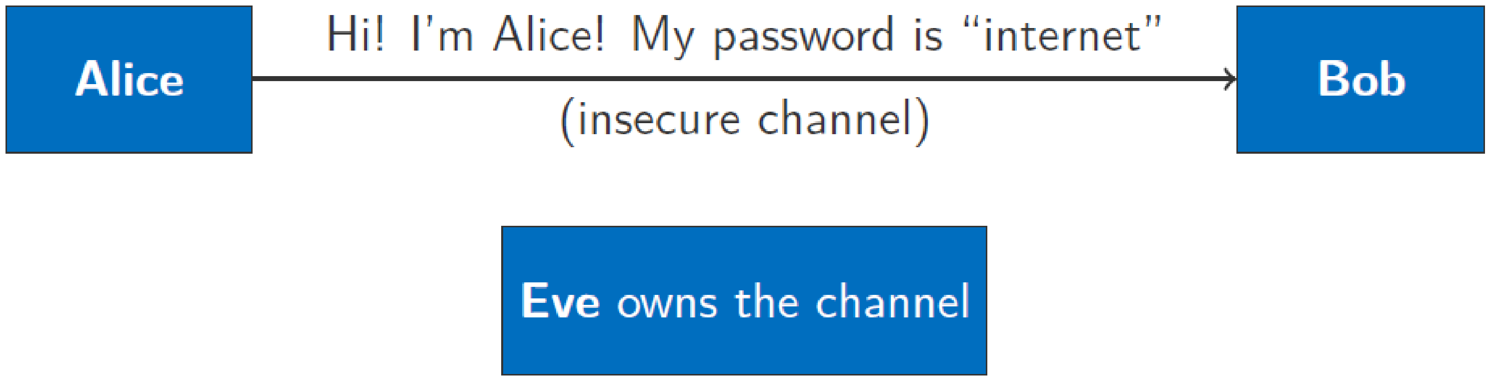
THE UNIVERSITY OF SYDNEY

Authentication Attacks

- **Impersonation**
- **Relay**
- **Interleaving**: impersonation involving selective combination of information from one or more previous or concurrent sessions.
- **Reflection:** an interleaving attack involving sending information from an ongoing authentication session back to the originator.
- **Forced Delay:** adversary intercepts a message and relays it at some later point in time (not the same as replay!)
- **Chosen Text:** attack on challenge-response where an adversary chooses challenges in an attempt to extract the secret key.

Passwords

- Passwords are a simple (and weak) method of authentication.
  - 1. Alice creates a secret password and shares it only with Bob, at some other point in time.
  - 2. To authenticate, Alice reveals her name and password.
- Passwords are usually stored hashed on the server, for extra security. If the server is compromised, these hashes need to be cracked to reveal the password.



Problems with Passwords

- Passwords can be eavesdropped and replayed.
  - Partially fixed by securing the channel using e.g. Diffie–Hellman and symmetric crypto.
  - Still vulnerable to keyloggers, device theft, ...

- Passwords are usually drawn from a small keyspace, or re-used because they're hard to remember.
  - Many sites still limit passwords to 8, 12, 16 characters.
  - Dictionary attacks are very possible.
  - One site compromised ⇒ every site using that password compromised.
  - People are basically bad at making up passwords that can't get machine-cracked.

Dictionary Attacks

- Since passwords are hard to make up and remember, most human-generated passwords can be found in a dictionary.
  - Pre-compute all password hashes for the dictionary.
  - On receiving a password hash, look it up in the precomputed table.
  - If it exists, the password is now known.

- Dictionary attack using massive password tables are easily re-usable and shareable.

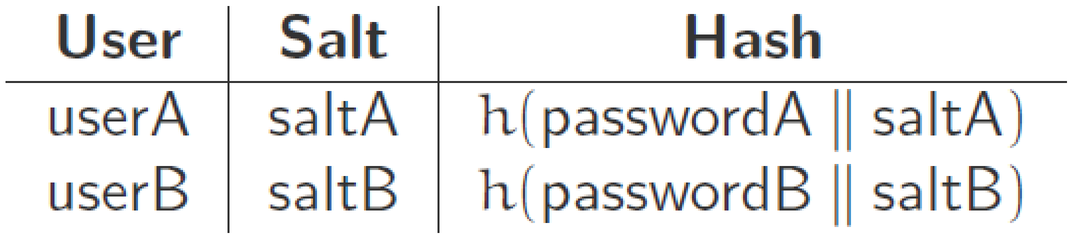
- Major extension to this attack: Rainbow tables use very compact storage.

Salting Passwords

- For a password p and hash function h, rather than using h(p) as the password hash, generate a random string s and use h(p || s). The string s is called the salt.
  - Salts are usually stored right next to the password hashes.
  - Salt is chosen at random.

- This thwarts dictionary attacks that rely on massive precomputation. If the salt is known, however, brute-force dictionary attacks may be run.

- UNIX Passwords use a Public Salt like this:



User	Salt	Hash
userA	saltA	h(passwordA || saltA)
userB	saltB	h(passwordB || saltB)

UNIX /etc/passwd

- Old UNIX passwords use DES as a hash function:
  - 1. Truncate password to 8 characters (7 bits/char ⇒ 56 bits)
  - 2. Encrypt a 64-bit block of 0's using truncated password as key.
  - 3. Output is fed back as input for a total of 25 times.
  - 4. A 2-byte salt is used to modify the expansion function, preventing the use of standard DES chips for cracking.

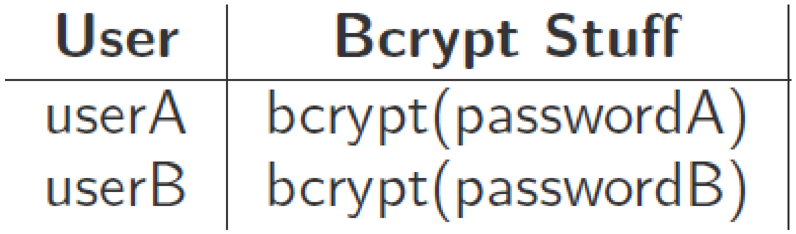
- If a login name of nick had a salt of wN, this might generate a line in /etc/passwd like:
  - user:password:uid:gid:homedir:shell
  - nick:wNX1CiVBBfQCk:1001:1001:/home/nick:/bin/sh
  - This hash was visible to all users of the system. (BAD!) Nowadays, the hashed passwords and salts are locked away in a separate file, and are not readable.

Brute Force Hashes

- Millions of passwords per second on CPUs or billions of passwords per second on GPUs.
- For a password composed of [a-zA-Z0-9] and symbols, hashed with SHA256, the Antminer S9 can break a password in an average of:
  - 8 chars: 1 days
  - 10 chars: 25 days
- Brute forcing password hashes is easily parallelised: N machines give a factor of N speedup.
- Standard hashing algorithms (MD5, SHA1, SHA256) are not good enough for passwords.
  - Hashing algorithms were designed to run fast.
  - Password hashes should ideally be slow to slow down brute-force attacks.
  - Brute forcing these algorithms is trivial even with a salt.

Modern password hashing: bcrypt

- bcrypt is a key derivation function for passwords — highly suggested for all websites.
  - Simple and clean implementations for many languages (no excuse to not use it, or roll your own hashing scheme)
  - Salts are handled automatically: developer doesn't even need to know they exist.
  - Key stretching: perform 2^c iterations of a hash, where c is a tunable cost parameter. Over time, c can be increased to make it slower.



User	Bcrypt Stuff
userA	bcrypt(passwordA)
userB	bcrypt(passwordB)

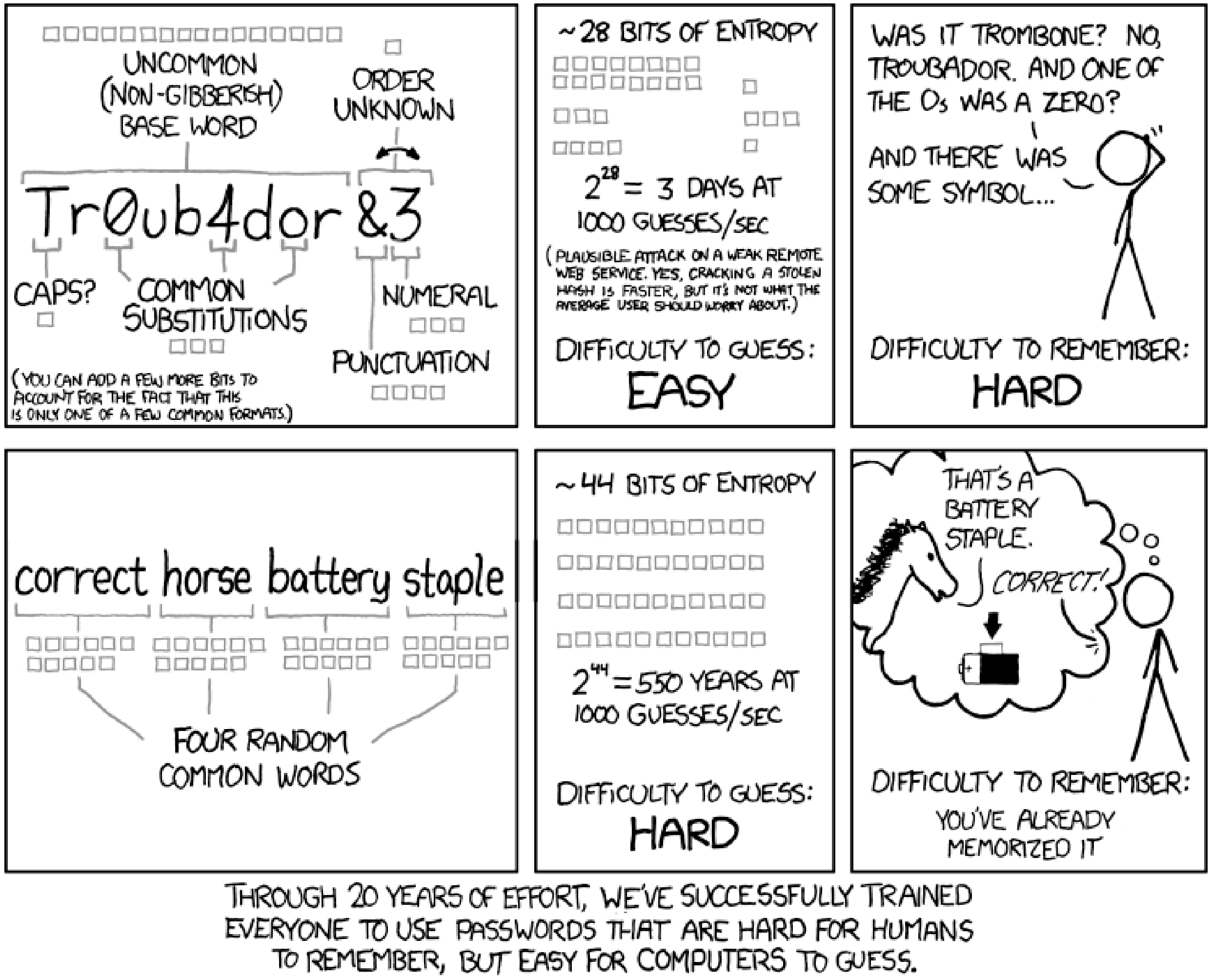
Modern password hashing: scrypt

- bcrypt aims to make hashing more expensive by using more time. It is still vulnerable to hardware attacks, since iterated hashes can be implemented in hardware.

- scrypt aims to make password hashing harder by using more space. It makes hardware implementations difficult by using vast amounts of memory.
  - 1. Generate a large vector of pseudorandom bit-strings.
  - 2. Password derivation function performs random lookups into this vector.
  - 3. Straightforward implementation requires entire vector to remain in memory.
  - 4. Complex implementation recreates vector elements (time/space tradeoff).

- Core algorithm used in Litecoin (cryptocurrency like Bitcoin) to discourage hardware-based mining implementations.

Problems with Passwords



UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

https://xkcd.com/936/

HTTP Authentication

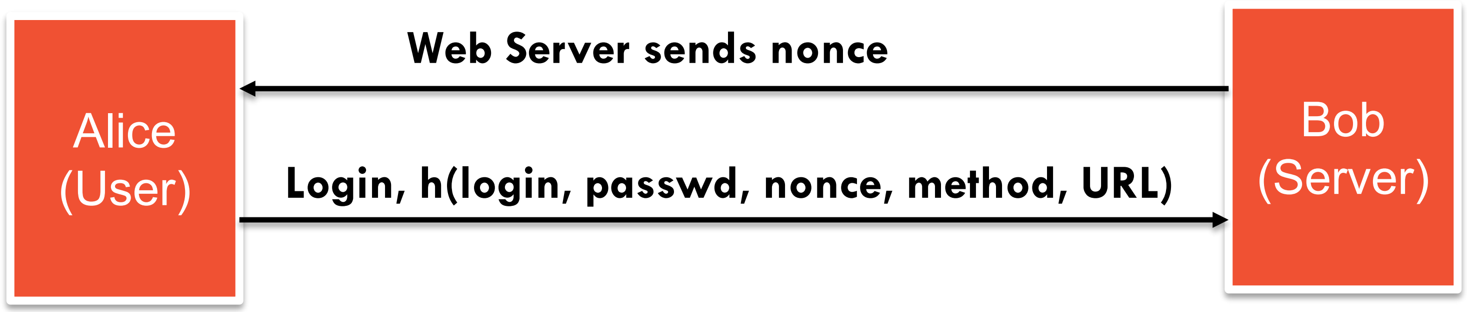
THE UNIVERSITY OF SYDNEY

Naïve HTTP (web server) Authentication

- Basic Authentication:
  - Access is segregated by realms.
  - Simple base64 encoding of username:password.

    WWW-Authenticate: Basic realm="Control Panel"

    Authentication: Basic QWRtaW46Zm9vYmFy
- Digest Authentication
  - MD5 is used as the hash function.



One Time Passwords

- In a one-time password scheme, each password is used only once, e.g., foil eavesdroppers and replay attacks.
- Many variations:
  - Shared list of one-time-passwords.
  - Challenge/response table.
  - Sequentially update one-time passwords. (e.g. user creates and uploads k_{i+1} when using k_i)
  - One-time sequences based on a one-way function. (e.g. Lamport's one-time password scheme)

Lamport's One Time Passwords

- Setup: (generates a scheme for a maximum of n uses)
  - 1. Alice picks a random key k and computes a hash chain

$$w = h^n(k) = \overbrace{h(h(\cdots h(k)\cdots))}^{n \text{ times}}$$

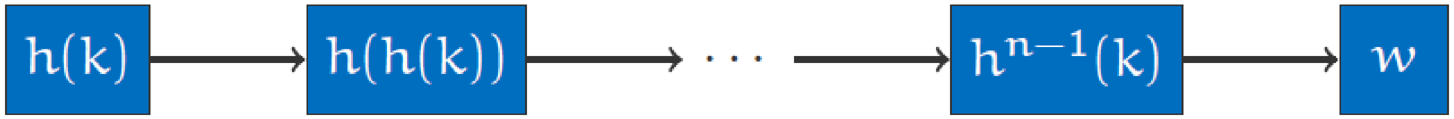
  - 2. Alice sends w to the server and sets the count $c = n - 1$.

- Authentication:
  - 1. Alice sends $x = h^c(k)$ to the server and decrements the count c.
  - 2. The server verifies that $h(x) = w$ and resets w to x.

Lamport's One Time Passwords

- Advantages
  - 1. No secrets stored on the server.
  - 2. Prevents replay attacks from eavesdropping.

- Disadvantages:
  - 1. A limited number of authentications before a new hash chain is set up.
  - 2. Vulnerable to a pre-play attack if the original secret is compromised.


$$\begin{tikzcd} \boxed{h(k)} \arrow[r] & \boxed{h(h(k))} \arrow[r] & \cdots \arrow[r] & \boxed{h^{n-1}(k)} \arrow[r] & \boxed{w} \end{tikzcd}$$

HOTP

- HOTP: An HMAC-Based One-Time Password Algorithm is defined in (RFC 4226) and used in end-user products.
- Setup:
  - 1. The client and server agree on a large ($\geqslant$ 160 bits) secret key k.
  - 2. The client and the server synchronise an 8-byte counter c, which increases over time.
- Authentication:
  - 1. Define $\mathrm{HOTP}(k, c) = \mathrm{HMAC{-}SHA{-}1}(k, c) \bmod 10^d$.
  - 2. The client calculates $w = \mathrm{HOTP}(k, c)$ and transmits it to the server.
  - 3. The server verifies that w is $\mathrm{HOTP}(k, c)$ for the current value of c.
    - The mod 10^d just returns the lowest d decimal digits of the HMAC.
    - Authentication usually also allows a small “window of error” of c values, for users being slow at typing, or unsynchronised clocks.

TOTP

- TOTP: Time-Based One-Time Password Algorithm is defined in RFC 6238, and used in end-user products. It is an extension of HOTP.
  - Specifies that the counter c in HOTP should be
    $$(\text{Current time } - T_0)/X$$
    where T_0 and X are some pre-agreed number of seconds.
  - For example, $X = 30$ seconds would make the password change every 30 seconds.
  - T_0 may be when the user registered, or Unix time 0.

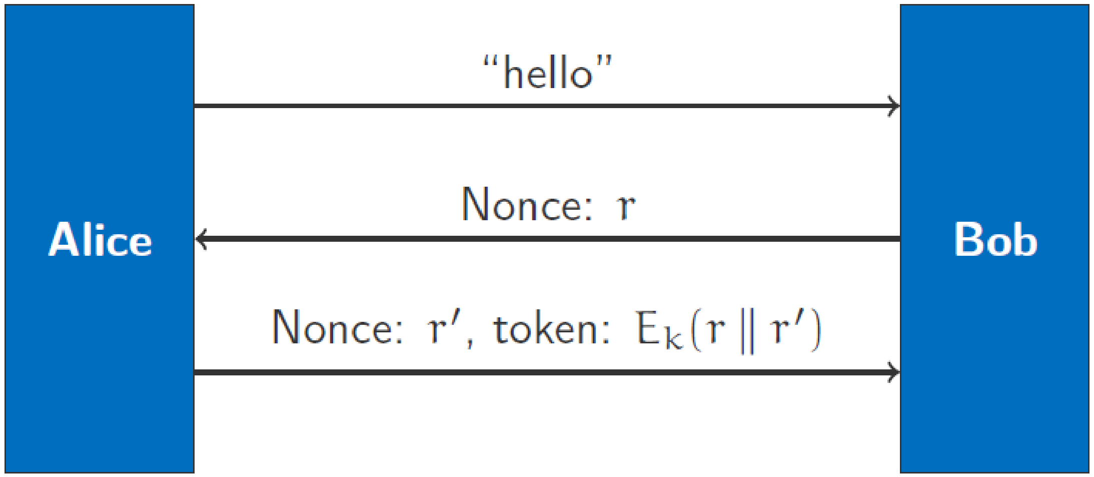
Challenge-Response Authentication

- One entity proves its identity to another by demonstrating knowledge of a secret, without revealing the secret itself
  - Done by providing a response to a time-variant challenge.
  - Response is dependent on both the challenge and the secret.

- Time-variant parameters are essential to counter replay and interleaving attacks, to provide uniqueness and timeliness guarantees, and to prevent certain chosen-ciphertext attacks. Some examples of time-variant parameters:
  - Nonces
  - Sequence Numbers
  - Timestamps

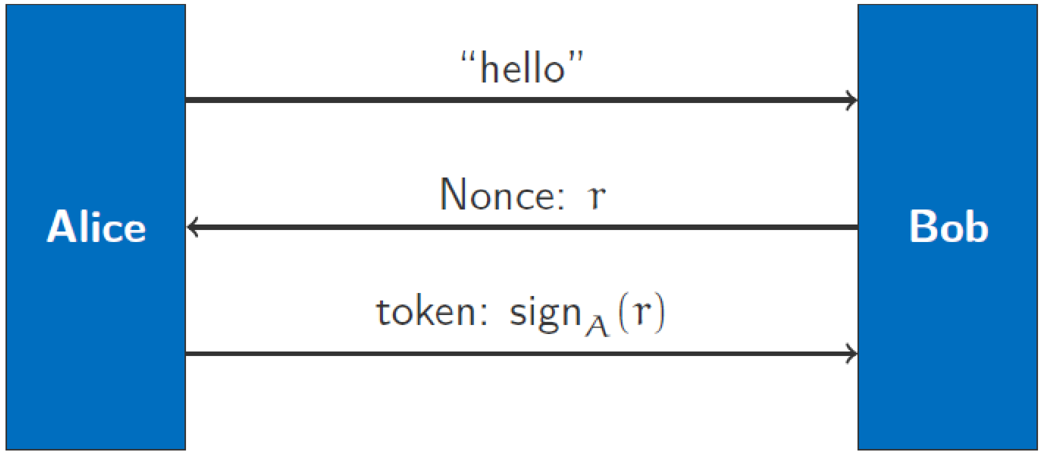
Challenge-Response Authentication using Symmetric Crypto

- Bob and Alice have a shared secret k and have agreed on some keyed encryption or hash algorithm E_k.
  - 1. Alice initiates communications with Bob.
  - 2. Bob generates a nonce r and sends it to Alice.
  - 3. Alice generates a nonce r' and sends Bob r' and $E_k(r \parallel r')$.
  - 4. Bob computes $E_k(r \parallel r')$ and compares with what Alice sent.



Challenge-Response Authentication using Asymmetric Crypto

- Alice publishes her public key A to the world, which Bob has a copy of (and verified its authenticity at some previous point in time).
  - 1. Alice initiates communications with Bob.
  - 2. Bob generates a nonce r and sends it to Alice.
  - 3. Alice signs the nonce $\text{sign}_A(r)$ and sends the result to Bob.
  - 4. Bob verifies the signature using Alice's public key.
- No shared secrets needed to be stored.



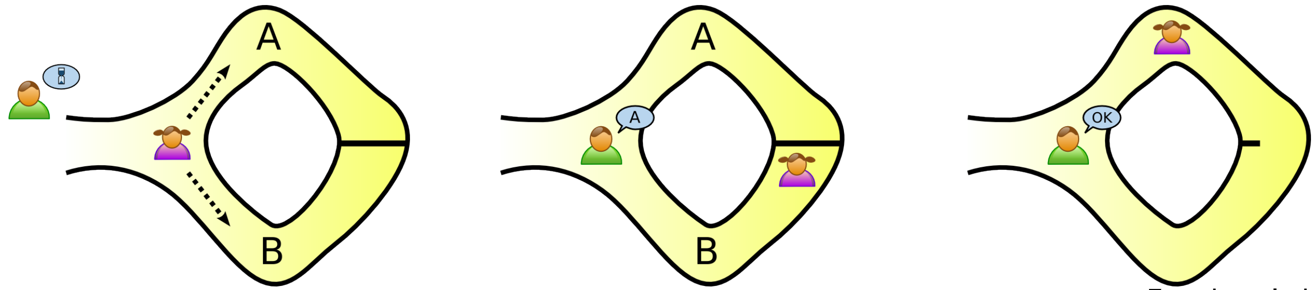
Zero Knowledge Proofs

- Zero-Knowledge Proofs (ZKP) are designed to allow a prover to demonstrate knowledge of a secret, while revealing no information at all about the secret.
  - ZKPs usually consist of many challenge-response rounds.
  - An adversary can cheat with a small probability on a single round.
  - The probability of cheating successfully should decrease exponentially (in a good ZKP protocol) in the number of interactive rounds.
  - For a large enough number of rounds, the probability of a successful cheater is effectively 0.

- Zero-Knowledge proof protocols are usually quite difficult to come up with but have applications in challenge-response authentication.

Zero Knowledge Proofs

– Ali Baba's Cave — Quisquater & Guillou (1989)
  – Ali Baba's cave contains a trapdoor, which only opens with a secret password.
  – Peggy, the prover, claims to know the password and wants to convince Victor, the verifier, that she does but doesn't want to tell Victor the password.
    - 1. Victor stands at the entrance while Peggy enters a random branch.
    - 2. Peggy stands next to the trapdoor while Victor enters the cave, and yells for Peggy to come out from A or B.
  – If Peggy knows the password, she can always succeed. If she does not, the probability of n successes is 2^{-n}.



Zero-knowledge proof - Wikipedia

Zero Knowledge Proofs

- Peggy cannot cheat Victor. The first time Peggy fails, Victor knows Peggy is not legitimate. Hence, each round is called an accreditation.

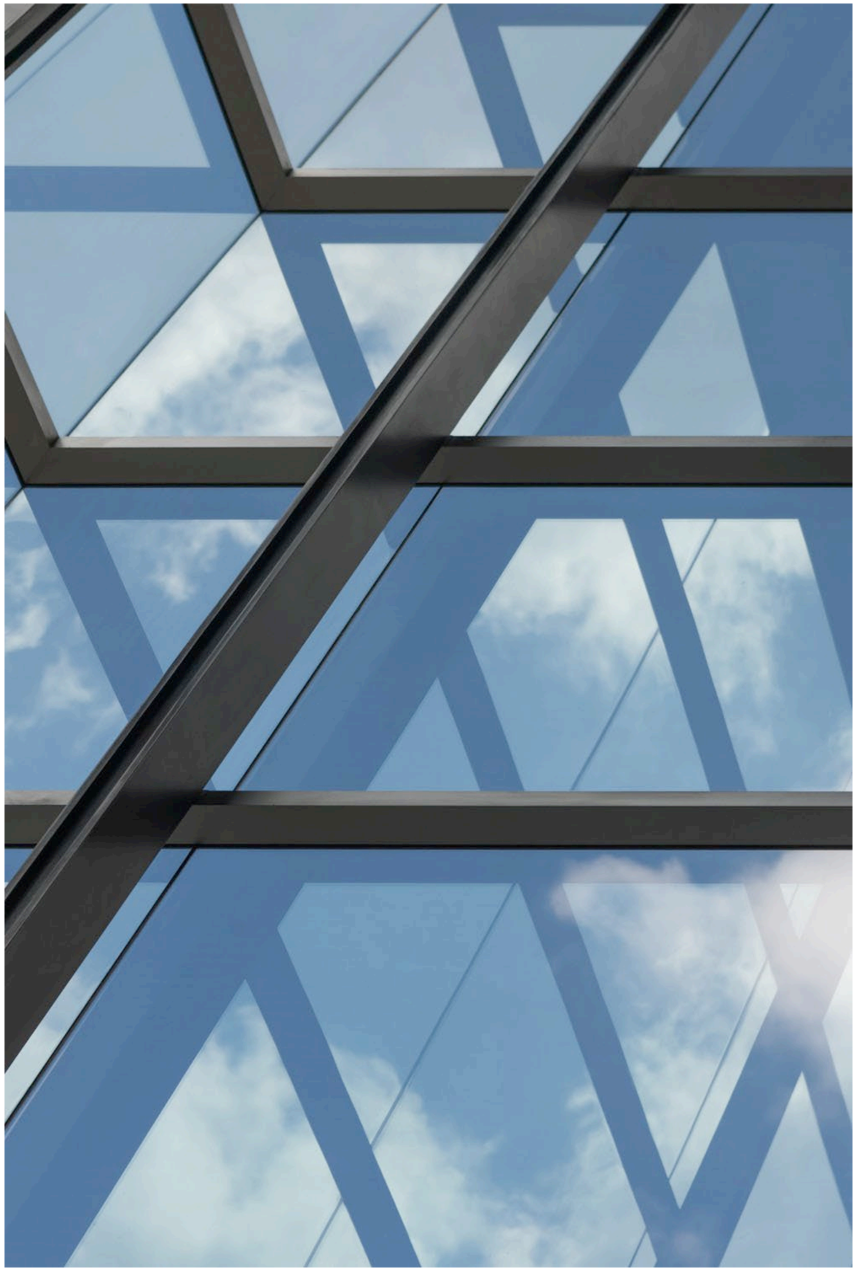
- With each accreditation, certainty gets better and better up to a desired level.

- Properties of ZKPs:
  - Victor cannot learn anything from the protocol, except that Peggy knows something.
  - Peggy cannot cheat Victor.
  - Victor cannot pretend to be Peggy to any third party.

- Dining cryptographers problem - Wikipedia

Questions

- What is the factoring attack in RSA and how can we mitigate this?
- What is a key security property provided by digital signatures that is not provided by message authentication codes (MACs)?
- What are some methods of improving password-based authentication?



THE UNIVERSITY OF
SYDNEY