Authentication

Luke Anderson

luke@lukeanderson.com.au

5th April 2019

University Of Sydney

Overview

Crypto-Bulletin

Crypto-Bulletin

Canadian Police Raid 'Orcus RAT' Author

https://krebsonsecurity.com/2019/04/canadian-police-raid-orcus-rat-author/

China's Kunlun in talks with US over Grindr: filing

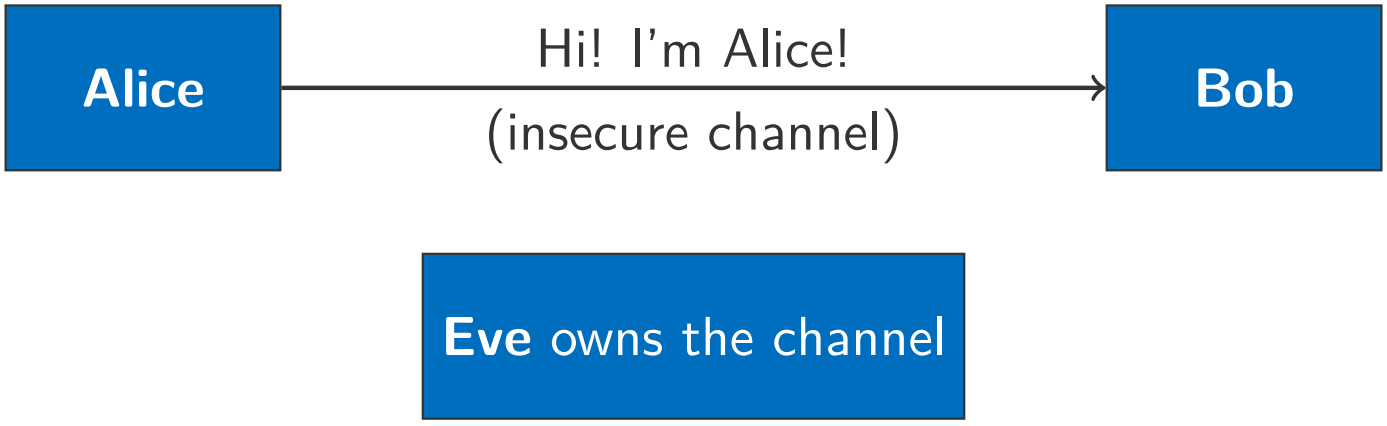
https://www.itnews.com.au/news/chinas-kunlun-in-talks-with-us-over-grindr-filing-523236

Google Warns of Growing Android Attack Vector: Backdoored SDKs and Pre-Installed Apps

https://threatpost.com/google-warns-of-growing-android-attack-vector-backdoored-sdks-and-pre-installed-apps/143332/

Authentication

Authentication



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

Authentication: Definition

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 cannot reuse an identification exchange with Alice so as to impersonate her in conversations with others.
- The probability that Eve can successfully impersonate Alice is 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, handheld password generator, 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:

- Key revocation.
- DNA and fingerprints are left everywhere.
- How do you give a mugger your fingerprint?
- How do you authenticate with a black eye?

Examples of Authentication

To verify identity as a precursor to communications:

- Letting people know the bomb threat really is from the IRA.

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 (swipecard)
- Border crossing (passport, (fingerprints too in the USA))

To facilitate resource tracking and billing:

- Mobile phone access

Attacks on Authentication

Attacks on Authentication

- **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.

Classic Attack on Authentication

In the late 1980s, the South African Defence Force (SADF) was fighting a war in northern Namibia and southern Angola with a goal to keep Namibia under white rule and impose UNITA as a client government. During this conflict, the Cubans broke the South African Air Force (SAAF) identify-friend-or-foe (IFF) system by performing a man-in-the-middle attack.

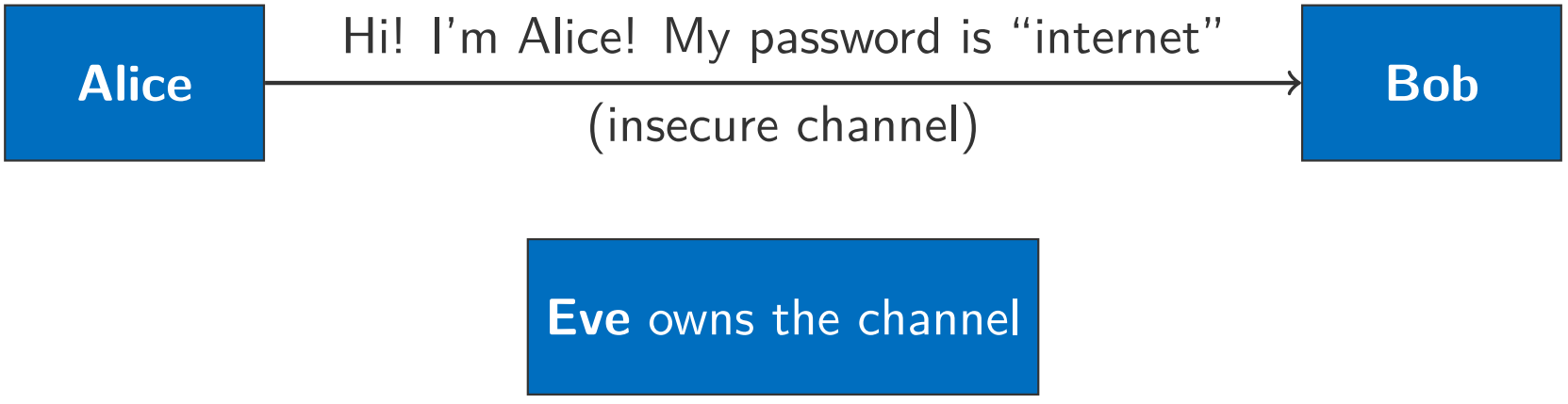
- Cubans waited until SAAF bombers raided a target in Angola
- Cubans then sent MIGs directly into SA air space in Namibia
- SAAF air defence queries MIGs using IFF
- MIGs relay signal to Angolan air defence batteries
- Angolan batteries bounce the IFF challenge of the SAAF bombers and then relayed back to MIGs in real-time

SADF casualties were proof that air supremacy was lost, and a factor in abandoning Namibia (and a step to majority rule in South Africa)

Passwords

Passwords

Passwords are a simple (and weak) method of authentication.



1. Bob creates a secret *password* and shares it only with Alice, at some other point in time.
2. To authenticate, Bob reveals his 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.

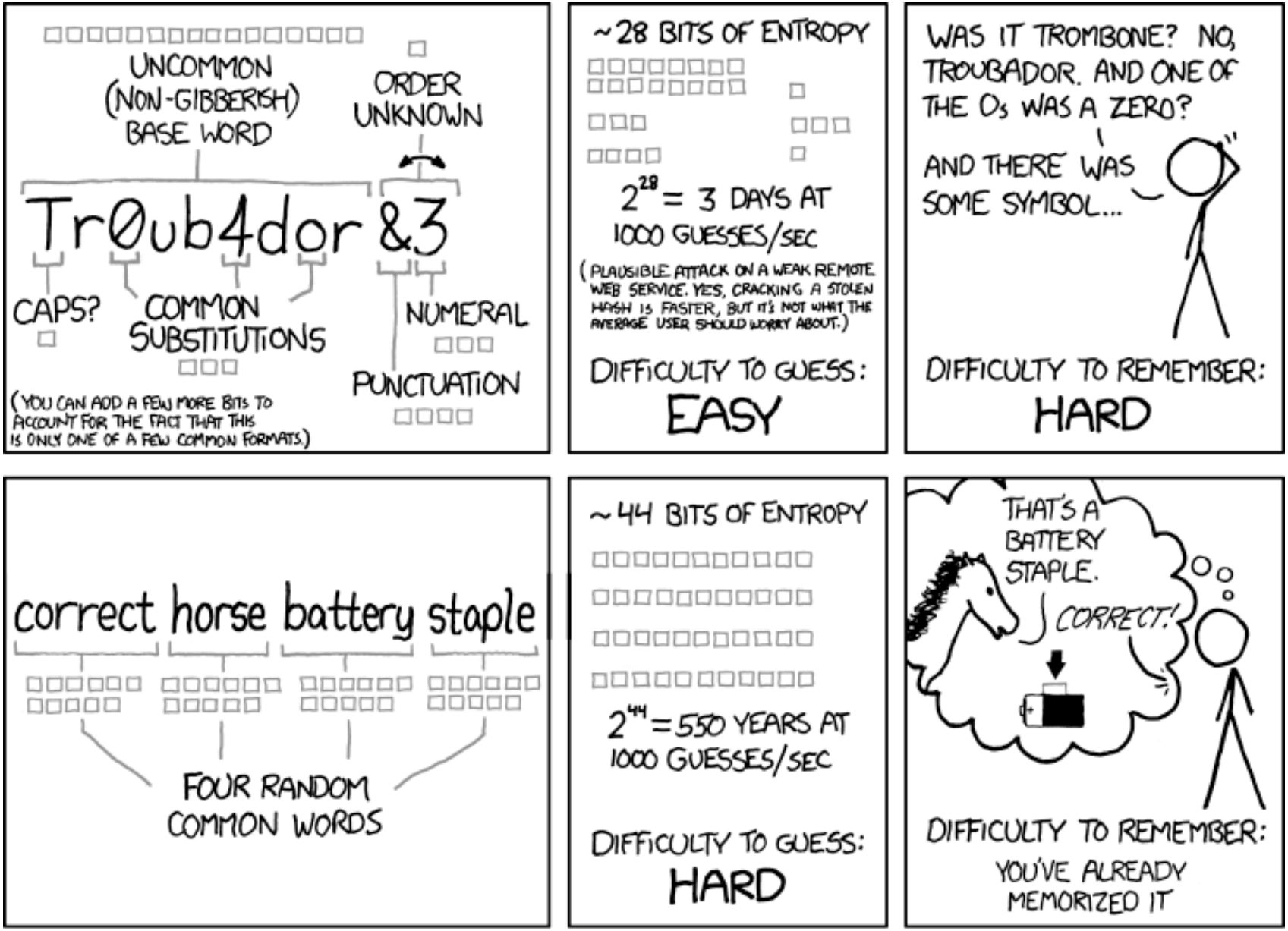
Problems with Passwords

Passwords can be eavesdropped and replayed.

- Partially fixed by securing the channel using e.g. Diffie–Hellman and symmetric crypto.
- Still vulnerable to keyloggers, nosy housemates, ...

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 $\implies$ every site using that password compromised.
- People are basically bad at making up passwords that can't get machine-cracked.



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/

UNIX /etc/passwd

Old[1] UNIX passwords use DES as a hash function:

1. Truncate password to 8 characters (7 bits/char $\implies$ 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.

[1] Such as 7th edition (1979): see crypt(3) and passwd(5)

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.

Advantages:

- Re-usable and shareable.
- Won't work on all users, but probably a large portion.
- *Major extension* to this attack: Rainbow tables use very compact storage.

Salting passwords

For a password p and hash function h, rather than using $\mathrm{h}(\mathrm{p})$ as the password hash, generate a random string s and use $\mathrm{h}(\mathrm{p} \,\|\, s)$. The string s is called the *salt*.

- Salts are usually stored right next to the password hashes.
  - UNIX Passwords use a Public Salt like this.
  - Salt is chosen at random.
- This thwarts dictionary attacks that rely on massive precomputation.
- Once the salt is known, however, brute-force dictionary attacks may be run.

User	**Salt**	**Hash**
userA	saltA	$\mathrm{h}(\text{passwordA} \,\|\, \text{saltA})$
userB	saltB	$\mathrm{h}(\text{passwordB} \,\|\, \text{saltB})$

Table: An example database table that handles password hashing and salting itself.

Brute Forcing Hashes

Millions of passwords per second on CPUs.
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 *embarrassingly parallel*: 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.

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

Table: An example database table that uses bcrypt. (Bcrypt takes care of salting and key stretching automatically)

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 are relatively easy to implement 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.

HTTP Authentication (Naïve)

Naive HTTP (web server) Authentication

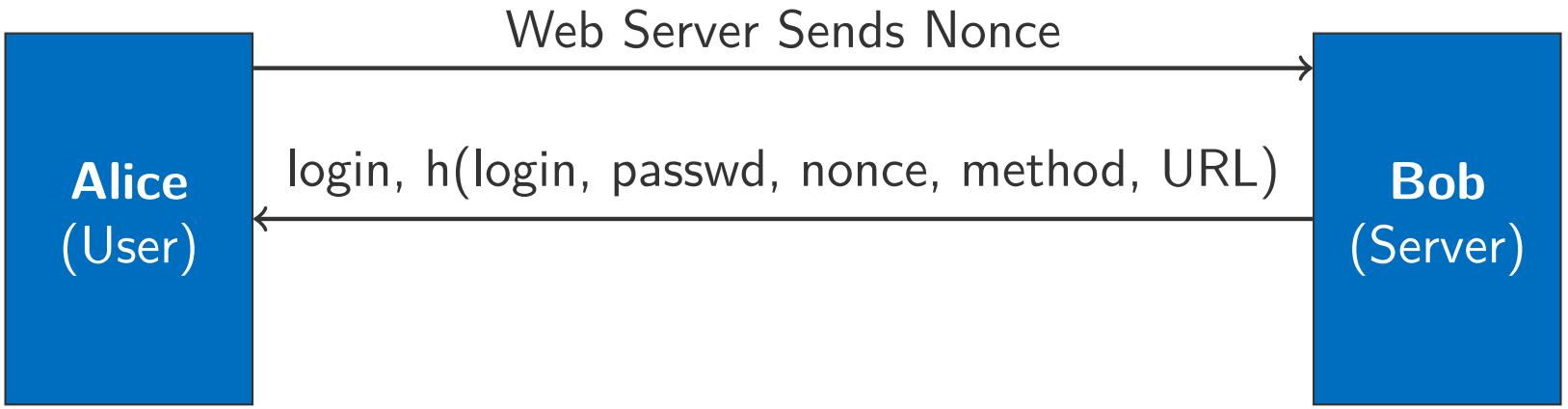
Basic Authentication:

- 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

One Time Passwords

In a *one time password* scheme, each password is used only once, in an attempt to 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

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

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.

HOTP

HOTP: An HMAC-Based One-Time Password Algorithm is defined in (RFC 4226), and used in end-user products such as Google Authenticator.

Setup:

1. The client and server agree on a large ($\geqslant 160$ bits) secret key k.
2. The client and the server synchronise a 8-byte counter c, which increases over time.

Authentication:

1. Define $\mathrm{HOTP}(\mathrm{k}, \mathrm{c}) = \text{HMAC-SHA-1}(\mathrm{k}, \mathrm{c}) \mod 10^{\mathrm{d}}$.
2. The client calculates $w = \mathrm{HOTP}(\mathrm{k}, \mathrm{c})$ and transmits it to the server.
3. The server verifies that w is $\mathrm{HOTP}(\mathrm{k}, \mathrm{c})$ 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 such as Google Authenticator. It is an extension of HOTP.

- Specifies that the counter c in the previous algorithm 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 0.

Challenge Response

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.

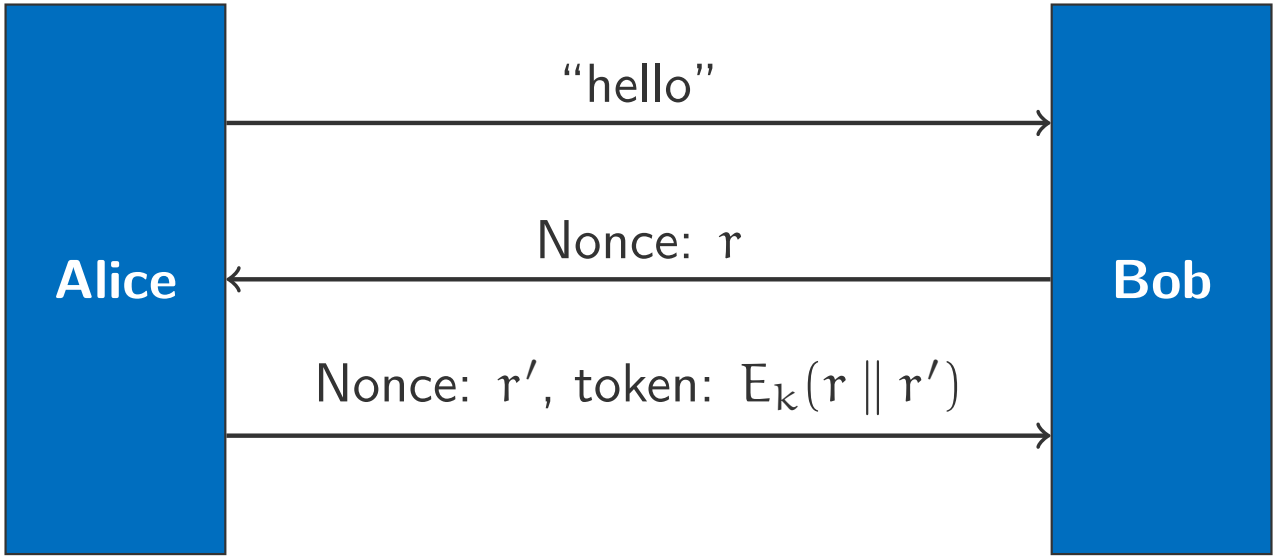
Time-variant parameters are essential to counter replay and interleaving attacks, to provide uniqueness and timeliness guarantees (e.g. freshness), and to prevent certain chosen-ciphertext attacks. Some examples of time-variant parameters:

- 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.

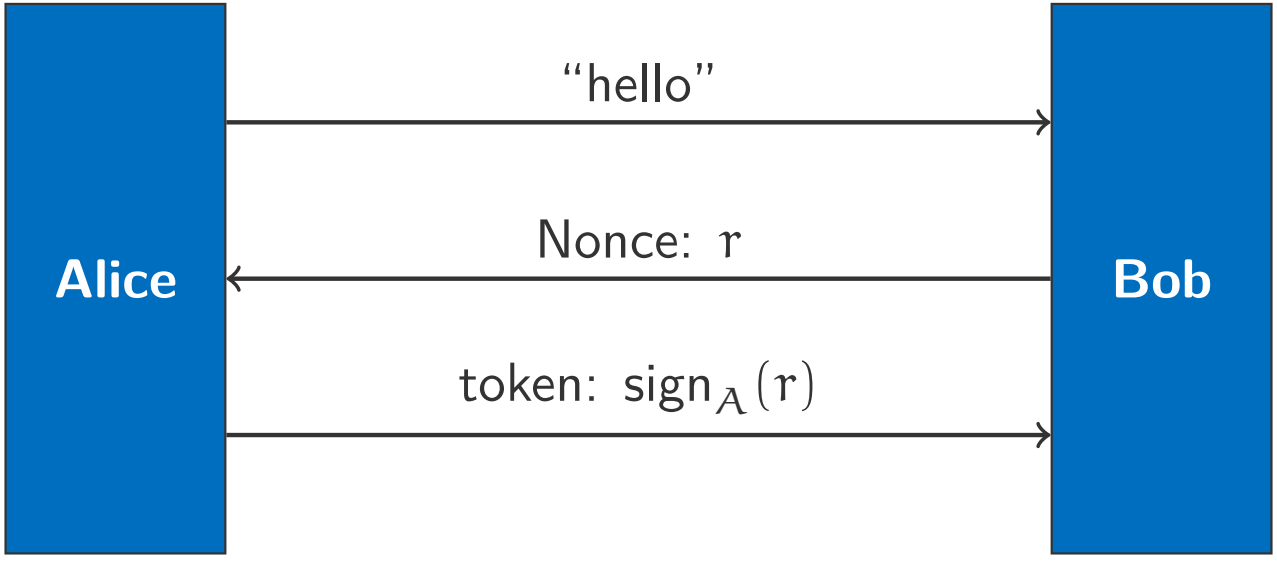
1. Alice initiates communications with Bob.
2. Bob generates a nonce r and sends it to Alice.
3. Alice generates a nonce r' and sends Bob r' and $E_k(r \parallel r')$.
4. Bob verifies $E_k(r \parallel r')$ is what Alice sent.



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 secrets needed to be stored on the server.



Zero-Knowledge Proofs

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.

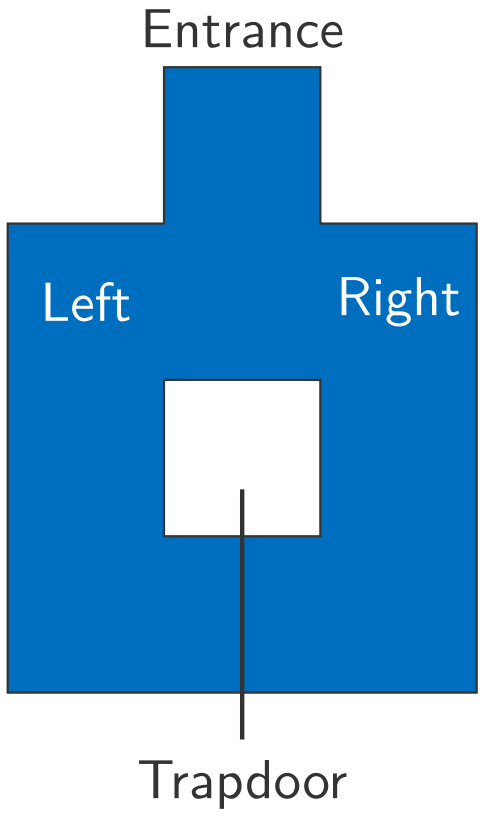
Zero-Knowledge proof protocols are usually quite difficult to come up with, but have applications in challenge-response authentication.

Ali Baba's Cave — Quisquater & Guillou (1989)

Ali Baba's cave contains a trapdoor, which only opens with a secret password. Peggy claims to know the password and wants to convince Victor that she does, but doesn't want to tell Victor the password.

1. Victor stands at the entrance while Peggy enters a random branch and stands next to the trapdoor..
2. Victor enters the cave to the fork, and yells for Peggy to come out the left or right.

If Peggy knows the password, she can always succeed. If she does not, the probability of n successes is 2^{-n}.



Zero-Knowledge Proofs

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

The Dining Cryptographers

The Dining Cryptographers Problem

Three cryptographers are sitting down to dinner at their favourite three-star restaurant. The waiting staff informs them that arrangements have been made with the maître d'hôtel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been the NSA. The three cryptographers respect each other's right to make an anonymous payment, but they wonder if the NSA is actually paying.

— David Chaum (1988)

- There is an easy solution involving a trusted third-party: call a member of the waiting staff over, and each whisper in their ear.
- Can it be done without a trusted third-party, assuming all cryptographers trust each other?

The Dining Cryptographers: Algorithm

The algorithm:

1. In secret, each cryptographer flips an unbiased coin, and reveals it to the cryptographer on their right.
2. Now there is a coin "shared" between each adjacent pair of cryptographers.
3. Each cryptographer states whether the two coins they can see are the same, or different.
4. **Except** if that cryptographer actually paid, they say the opposite.
5. An odd number of differences means that a cryptographer has paid, an even number means the NSA has paid.
6. This extends to any number of diners.

Two claims:

A. This actually works.
B. No single cryptographer can deduce any information about the others.

The Dining Cryptographers: Proof

- Let heads be 0 and tails be 1.
- When XORing two coins together, the result will be 0 if and only if they showed the same face.
- To flip an answer, report $1 \oplus x$ instead of x.

Let the faces of the three coins be a, b, c. The first cryptographer sees a and b, the second sees b and c, and the third sees c and a.

- If no-one flips their value,

$$(a \oplus b) \oplus (b \oplus c) \oplus (c \oplus a) = a \oplus a \oplus b \oplus b \oplus c \oplus c = 0$$

- If the second cryptographer flips their value,

$$(a \oplus b) \oplus (b \oplus c \oplus 1) \oplus (c \oplus a) = a \oplus a \oplus b \oplus b \oplus c \oplus c \oplus 1 = 1$$

- The case for one of the other two flipping their values is identical.

So, the algorithm is correct.

The Dining Cryptographers: Corollaries

- Shows unconditional secrecy channels can be used to construct an unconditional sender (and receiver) untraceability channel.
- Implies that a public-key distribution system can be used to construct a secure sender untraceability channel. (aka Anonymous Broadcast)

This protocol can be extended by having each pair share a longer one-time-pad rather than just a coin toss.

- Transfer many bytes at a time rather than just one bit.
- The result at the end is 0 or an anonymously broadcast message.
- Message can be mangled if multiple people broadcast at once (overcome using a random back-off procedure, similar to ethernet).