41900 – Fundamentals of Security

Authentication

Ashish Nanda

Ashish.Nanda@uts.edu.au

Authentication: Definition

Authentication is a means by which identity is established.

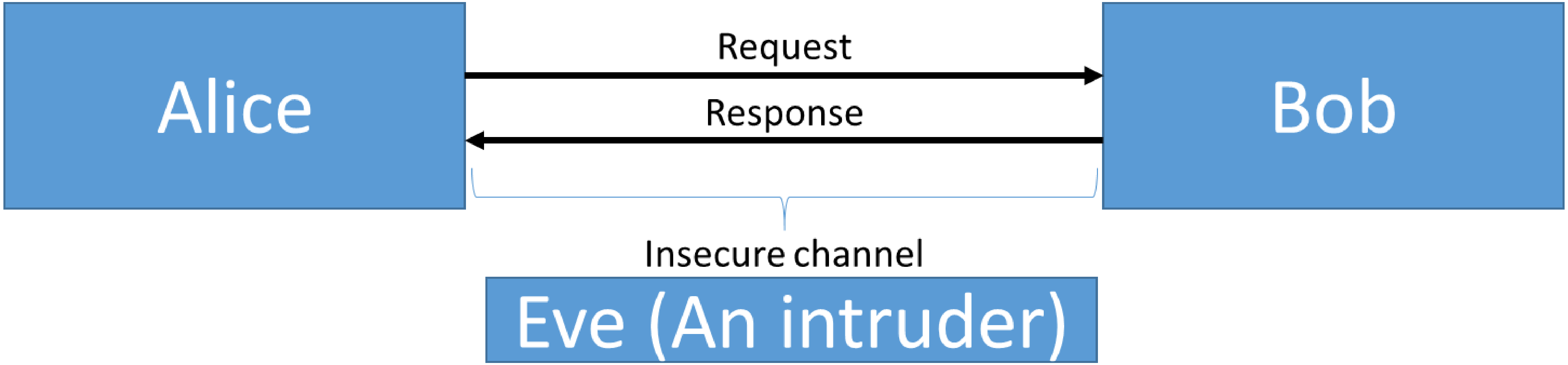
Challenge: Bob needs to ensure that the party at the end of the channel is Alice.

To do so, Bob must:

- Ensure Alice's identity.
- Ensure Alice has actively participated.

Aim: To achieve this over an **insecure channel** with an active attacker.

Authentication



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

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 (key card and PIN)
- Allow communications through a web server proxy
- Allow physical access to restricted areas (swipe card)
- Border crossing (passport, (fingerprints too in the USA))

To facilitate resource tracking and billing:

- Mobile phone access

Attacks on Authentication

Impersonation: impersonation involving selective combination of information from one or more previous or concurrent sessions.

Interleaving: 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 - Case Study

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.

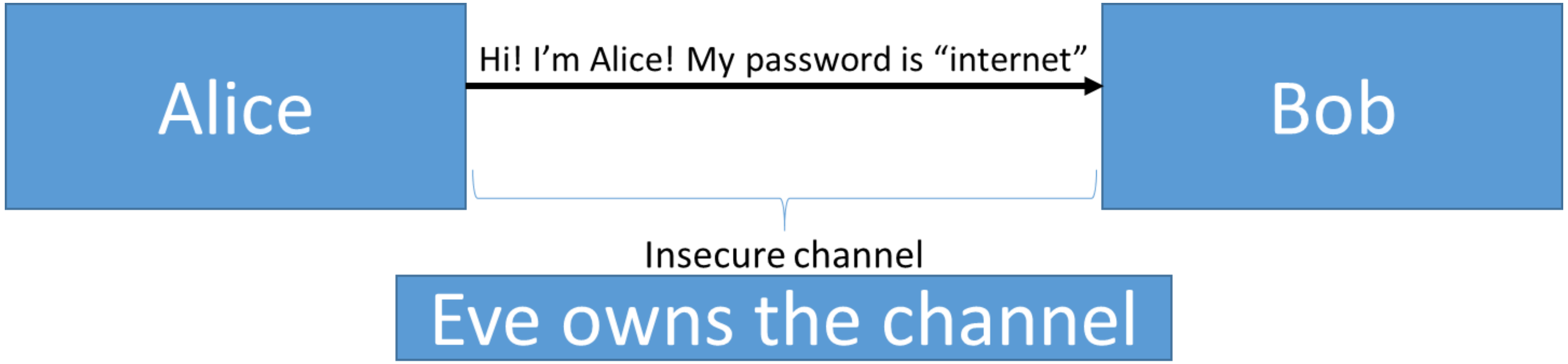
- Cubans waited until SAAF bombers raided a target in Angola
- Cubans then sent MIGs directly into SA air space in Namibia
- SAAF air defence queries MIGs using IFF
- MIGs relay signal to Angolan air defence batteries
- Angolan batteries bounce the IFF challenge of the SAAF bombers and then relayed back 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 are a simple (and weak) method of authentication.

- Bob creates a secret password and shares it only with Alice, at some other point in time.
- 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 key space, 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.

UNIX /etc/passwd

Old UNIX passwords use DES as a hash function:

- Truncate password to 8 characters **(7 bits/char ⇒ 56 bits)**
- Encrypt a **64-bit** block of **0**'s using truncated password as key.
- Output is fed back as input for a total of 25 times.
- 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.

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 **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.
    - 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	h(passwordA ‖ saltA)
userB	saltB	h(passwordB ‖ saltB)

Brute Forcing Hashes

Millions of passwords per second on CPUs.

Billions of passwords per second on GPUs.

For a password composed of [a-z; A-Z; 0-9] 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 **2c** iterations of a hash, where **c** is a tuneable cost parameter.

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.

- Generate a large vector of pseudorandom bit-strings.
- Password derivation function performs random lookups into this vector.
- Straightforward implementation requires entire vector to remain in memory.
- Complex implementation recreates vector elements (time/space tradeoff).

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

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)

- Alice picks a random key **k** and computes a hash chain

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

- Alice sends w to the server, and sets the count **c = n - 1**.

Authentication:

- Alice sends $\mathbf{x = h^c(k)}$ to the server, and decrements the count **c**.
- The server verifies that **h(x) = w**, and resets **w** to **x**.

Lamport's One Time Passwords

Advantages

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

Disadvantages:

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

$$\boxed{h(k)} \rightarrow \boxed{h\big(h(k)\big)} \xrightarrow{\quad \dots \quad} \boxed{h^{n-1}(k)} \rightarrow \boxed{w}$$

HOTP

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

Setup:

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

Authentication:

- Define **HOTP(k, c) = HMAC-SHA-1(k, c) mod 10^d**.
- The client calculates **w = HOTP(k, c)** and transmits it to the server.
- The server verifies that **w** is **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

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

(Current time - T_0)/X

- where $\mathbf{T_0}$ and **X** are some pre-agreed number of seconds.
- For example, **X = 30** seconds would make the password change every **30** seconds.
- $\mathbf{T_0}$ may be when the user registered, or **0**.

Challenge-Response Authentication

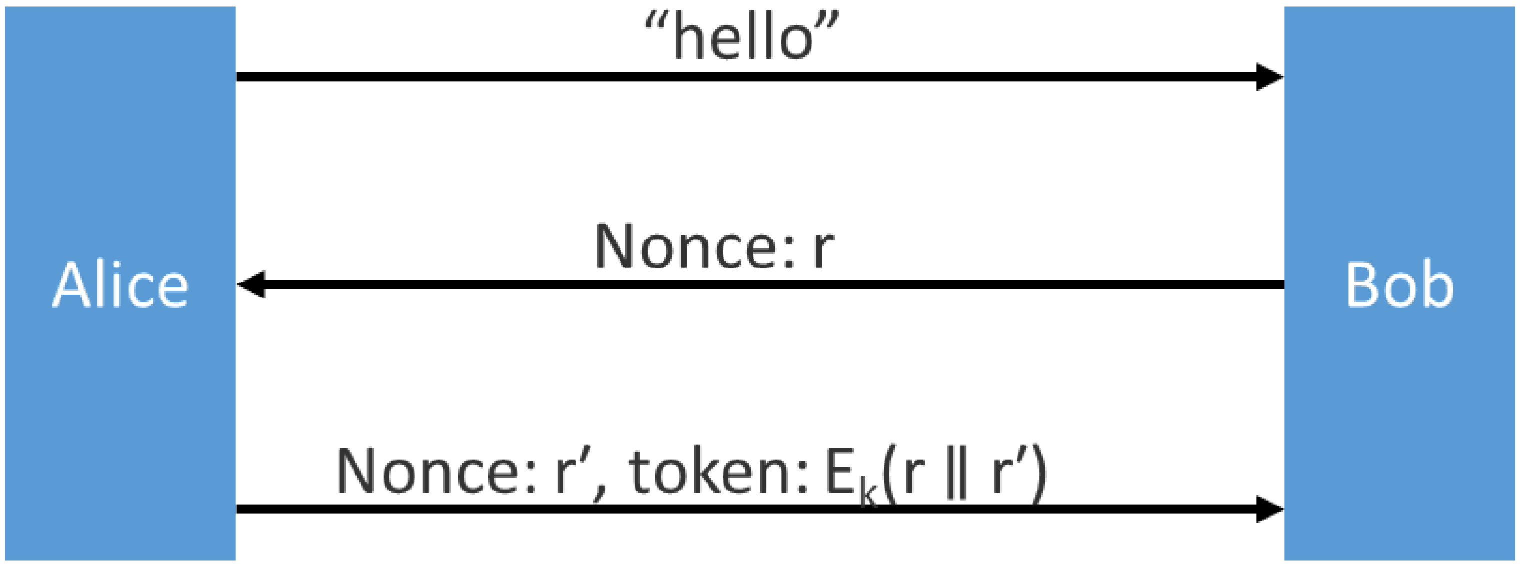
One entity proves its identity to another by demonstrating knowledge of a secret, without revealing the secret itself

- 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 (e.g. freshness), and to prevent certain chosen-ciphertext attacks. Some examples of time-variant parameters:

- Nonces
- Sequence Numbers
- Timestamps

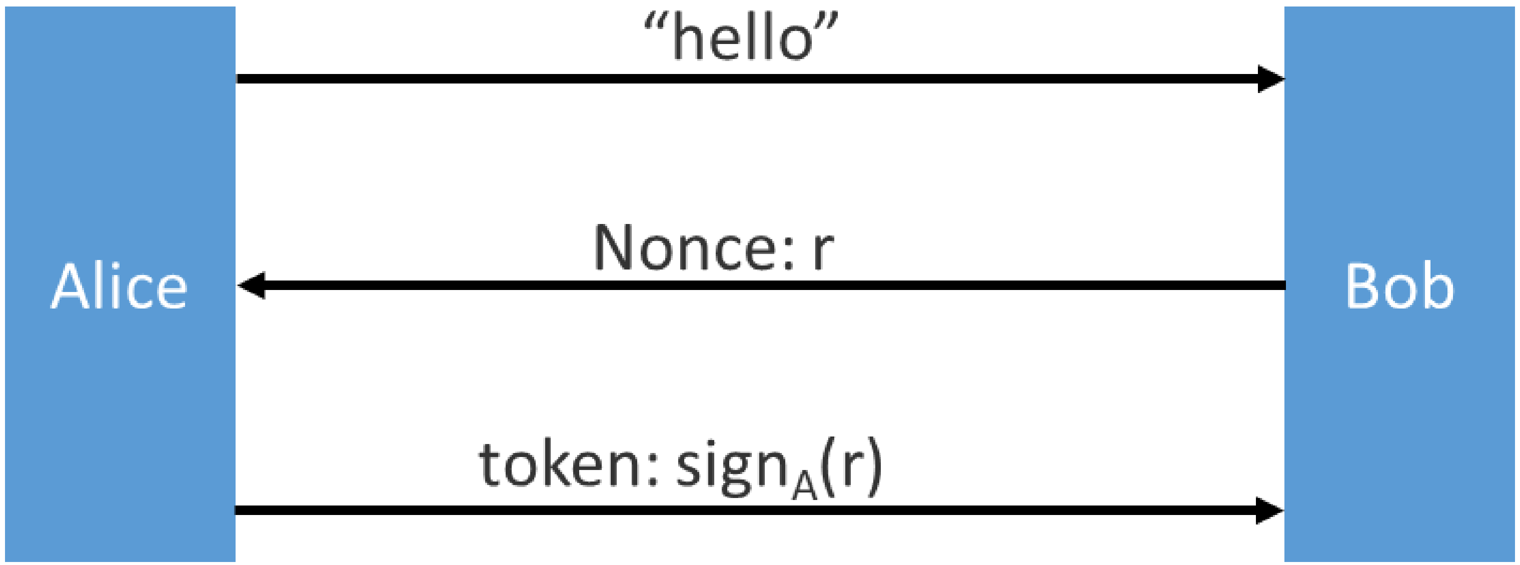
Challenge-Response Authentication using Symmetric Crypto



Bob and Alice have a shared secret **k**, and have agreed on some keyed encryption or hash algorithm $\mathbf{E_k}$.

- Alice initiates communications with Bob.
- Bob generates a nonce **r** and sends it to Alice.
- Alice generates a nonce **r′** and sends Bob **r′** and $\mathbf{E_k(r \| r')}$.
- Bob verifies $\mathbf{E_k(r \| r')}$ is 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).

- Alice initiates communications with Bob.
- Bob generates a nonce **r** and sends it to Alice.
- Alice signs the nonce **signA(r)**, and sends the result to Bob.
- Bob verifies the signature using Alice's public key.
- No secrets needed to be stored on the server.

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

Cut and Choose Protocol:

- Alice cuts something in half.
- Bob picks which half he wants.
- Alice takes the remaining half.

Each round is called an accreditation.

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.

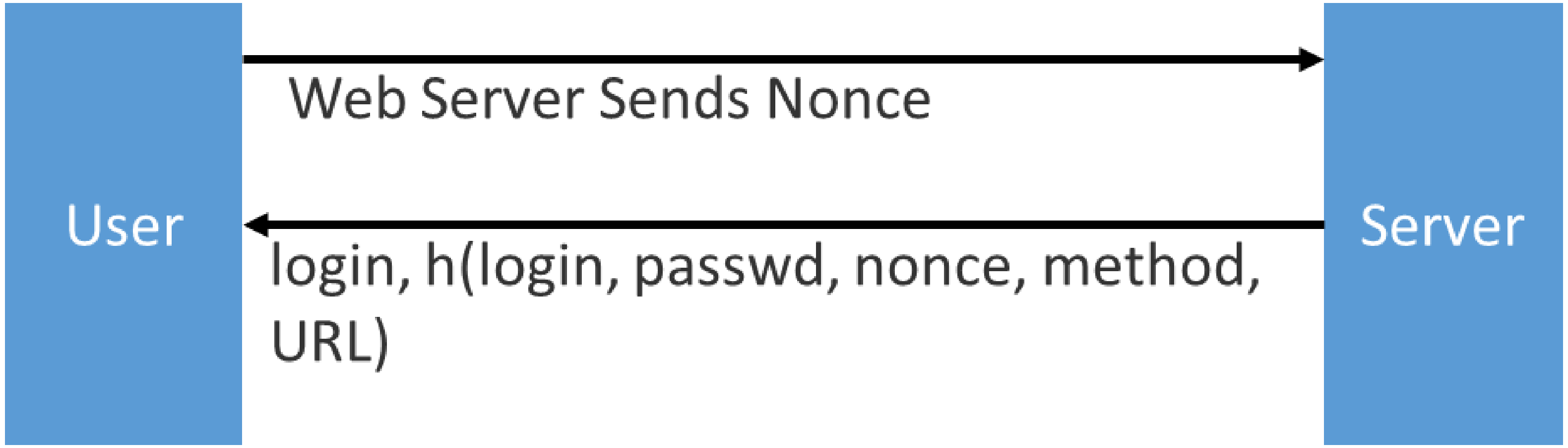
Naive HTTP (web server) Authentication

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.



Key Points

Authentication is not cryptography

- You have to consider system components

Passwords are here to stay

- They provide a basis for most forms of authentication

Protocols are important

- They can make masquerading harder

Authentication methods can be combined