COMPUTER & NETWORK SECURITY

Lecture 12:
Authentication

CRYPTO BULLETIN

- **Taking Down Fraud Sites is Whac-a-Mole**
  http://krebsonsecurity.com/2015/04/taking-down-fraud-sites-is-whac-a-mole/

- **'Massive vulnerability' uncovered in eBay Magento ecommerce system**
  http://www.itnews.com.au/News/402934,massive-vulnerability-uncovered-in-ebay-magento-ecommerce-system.aspx

AUTHENTICATION

How does Bob know that Alice is Alice, not Eve?

AUTHENTICATION

Authentication is a means by which **identity** is established

It allows one party to gain assurances about the identity of another party in a protocol, and that the second has actively participated.

The goal of authentication is to achieve all 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, A is able to successfully authenticate herself to Bob, i.e. Bob will complete the protocol having accepted 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 to Bob is negligible (e.g. computationally difficult).

All the above remain true even if Eve has seen many previous authentication sessions between Alice and Bob, has had experience in authenticating herself with both, and multiple authentication sessions are run simultaneously.

BASIS OF IDENTIFICATION

Something you know:
Passwords, PINs, secret keys, mother's maiden name

Something you have:
Magnetic cards, smart cards, physical keys, handheld password generators

Something you are:
biometrics (DNA, signatures, fingerprints, voice, retinal patterns, hand geometry, typing dialect/profiling)

Biometrics have major problems in real world situations:

- How do you revoke keys?
- Biology is messy – we leave traces (fingerprint, DNA, ...) of us everywhere
- How do you give a mugger your fingerprint?
- How do you authenticate if he's just hit you in the eye?

EXAMPLES OF AUTHENTICATION APPLICATIONS

To verify identity as 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 (e.g. password)
- withdrawal of money from an ATM (e.g. keycard, PIN)
- allow communications through a web server proxy
- allow physical access to restricted areas (e.g. swipecard)
- border crossings (e.g. passport)

To facilitate resource tracking & billing:

- mobile phone access

ATTACKS ON AUTHENTICATION

Impersonation

Relay

Interleaving:
impersonation involving selective combination of information from one or more previous or simultaneous 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 (note: 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.

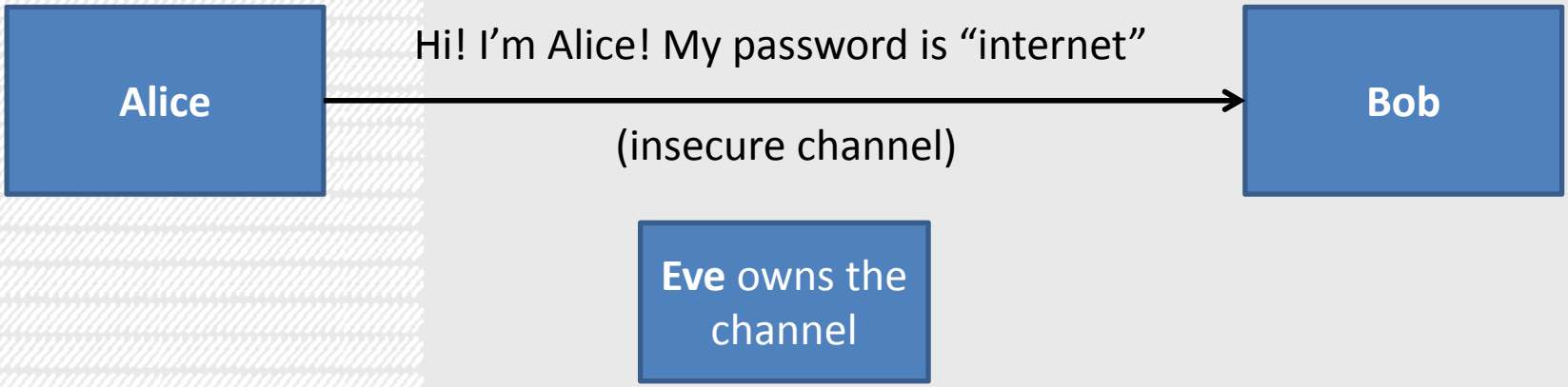
During this conflict, the Cubans broke the South African Air Force (SAAF) identify-friend-or-foe (IFF) system by performing a man-in-the-middle attack:

- 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 the 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 the simplest (and weakest) means of authentication



Password authentication is where a secret is shared between two parties. To authenticate, one party reveals their identity and their password.

Passwords are typically stored hashed on a server in a password file (so if the server is compromised, the passwords still needs to be cracked).

PASSWORDS HAVE MAJOR PROBLEMS

Passwords can be eavesdropped facilitates replay attacks

Passwords are reusable facilitates impersonation attacks by verifier

Passwords usually come from a small keyspace facilitates brute force attacks

Extremely low entropy

- English only has ~1.3bits/byte of real information
- dictionary attacks are possible

Humans are extremely poor random number generators

- makes dictionary attacks even easier (or unnecessary)

Humans are pathetic at remembering passwords and often reuse (or alternate between) old passwords

UNIX /ETC/PASSWD

Standard UNIX passwords use DES as a hash function

- password is truncated to 8 characters (@ 7 bits = 56 bits)
- then used as a key to encrypt a 64-bit block of 0's
- output is fed as input 25 times
- salt is used to modify the expansion function (32 -> 48 bits)
  - prevents use of standard DES chips to perform cracking

Standard UNIX format

user:password:uid:gid:gecos:homedir:shell
nick:wNX1CiVBBfQCk:1001:1001:nick:/home/nick:/bin/sh
LLLL:SSPPPPPPPPPP

Login Name	Salt	Password Hash
nick	wN	X1CiVBBfQCk

DICTIONARY ATTACKS

Humans don't like remembering long complex passwords

Humans don't generate random strings well either

Most passwords can be found in a dictionary

1. Pre-compute all password hashes for top 10^n words from the dictionary
2. On receiving a password hash, look it up in your pre-computed table
3. If it exists in the pre-computed table, you've broken the password

Advantages

- Reusable and shareable
- It won't work for all users but it will work for a significant portion
- Major extension to this attack (rainbow tables)
  = compact storage + finds *entirely random passwords*

SALTING PASSWORDS

Adding a t-bit salt to passwords strengthens them against dictionary and brute force attacks.

Public salt (e.g. UNIX passwords)

- salt is chosen at random
- an adversary must hash a guessed password p 2^t times to find if p is a valid password (when password cracking)
- prevents the use of dictionary attacks or rainbow tables

User	Salt	Password Hash
userA	saltA	h(passA | saltA)
userB	saltB	H(passB | saltB)

BRUTE FORCING HASHES

Millions of passwords per second on CPUs

Billions of passwords per second on GPUs

Password composed of [A-Za-z0-9] + symbols

- 8 char password estimated strength 2^{30}

You can break **8 char password hash** (SHA256) in **half an hour** on a modern CPU

- 10 char password estimated strength 2^{32}

You can break **10 char pass hash** (SHA256) in **less than two hours** on a modern CPU

Brute forcing password hashes is **trivially parallelizable** (N machines = N times faster)

Standard hashing algorithms (MD5, SHA256, ...) are **not good enough** for passwords

Brute forcing individual passwords is still 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 almost every language
(no excuse not to use it or to roll your own)

Salts are handled by bcrypt – developer doesn't even need to know they exist
(idiot proofing the algorithm for developers)

Performs key stretching to severely slow down the ability to brute force

key = H(**password**)
for i in range(2^k):
    key = H(**key** | **password**)

MODERN PASSWORD HASHING SCRYPT

bcrypt aims to make password hashing harder by using more computations

Still vulnerable to hardware attacks – relatively simple to implement in hardware

scrypt aims to make password hashing harder by using more resources

Makes hardware implementations difficult by using vast amounts of memory

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

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

ONE TIME PASSWORDS

Each password is only used once

(an attempt to foil eavesdroppers and replay attacks)

Many variations:

- Shared list of one-time passwords
- Challenge response table (set list of questions / answers)
- Sequentially updated one-time passwords
  (user creates and uploads key **i+1** when using key **i**)
- One time sequences based upon a one way function
  (e.g. Lamport's one-time password scheme)

LAMPORT'S ONE TIME PASSWORDS (S/KEY)

Setup:

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

$$\mathbf{w} = h^n(\mathbf{k}) = h(h(h(h(...h(\mathbf{k})))))$$

Alice sends single value **w** to the server

Alice sets count **c = n - 1**

Authentication:

Alice sends $\mathbf{x} = h^c(\mathbf{k})$ to the server

Alice sets count **c = c – 1**

The server verifies $h(\mathbf{x}) = \mathbf{w}$

The server sets **w = x**

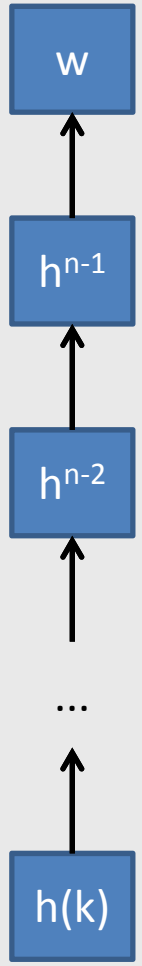
LAMPORT'S ONE TIME PASSWORDS (S/KEY)

Advantages:

- Prevents eavesdropping
- No secrets stored on server

Disadvantages:

- A limited number of authentications before a new hash chain must be set up
- Vulnerable to a pre-play attack if unused passwords are compromised



TIME-BASED + HMAC-BASED ONE TIME PASSWORD ALGORITHM (GOOGLE AUTHENTICATOR)

HMAC-based One-time Password Algorithm (RFC 4226)

Select a secret key **K** and an increasing counter value **C**

HOTP(**K**, **C**) = HMAC(**K**, **C**) mod 10^{d}

HMAC(**K**, **C**) = SHA (**K** $\oplus$ 0x5c5c… ‖ SHA (**K** $\oplus$ 0x3636… ‖ **C**))

(the mod 10^{d} allows the user to enter a **d** digit number)

Time-based One-time Password Algorithm (RFC 6238)

Replace increasing counter value **C** with an integer time step **T**

T = (Unix time – $\mathbf{T_0}$) / 30 [Unix time = secs since 1970]

$\mathbf{T_0}$ may be when the user registered or 0

Changes password every 30 seconds

Small issues with keeping reasonably synchronous time

CHALLENGE-RESPONSE AUTHENTICATION

One entity proves it's identity to another by demonstrating knowledge of a secret without revealing the secret itself

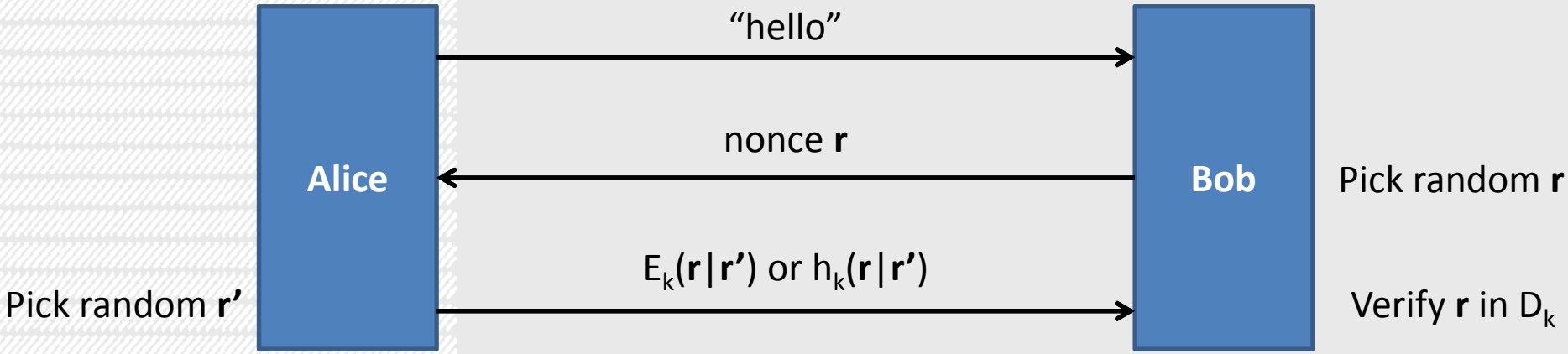
Done by providing a response to a time variant challenge, where the response is dependent on the challenge and the secret

Time variant parameters may be used to counter replay and interleaving attacks, to provide uniqueness or timeliness guarantees (e.g. freshness), and to prevent certain chosen-cyphertext attacks

- nonces
- sequence numbers (aka serial numbers, counters)
- timestamps

CHALLENGE-RESPONSE AUTHENTICATION USING SYMMETRIC TECHNIQUES

Symmetric cypher or MAC



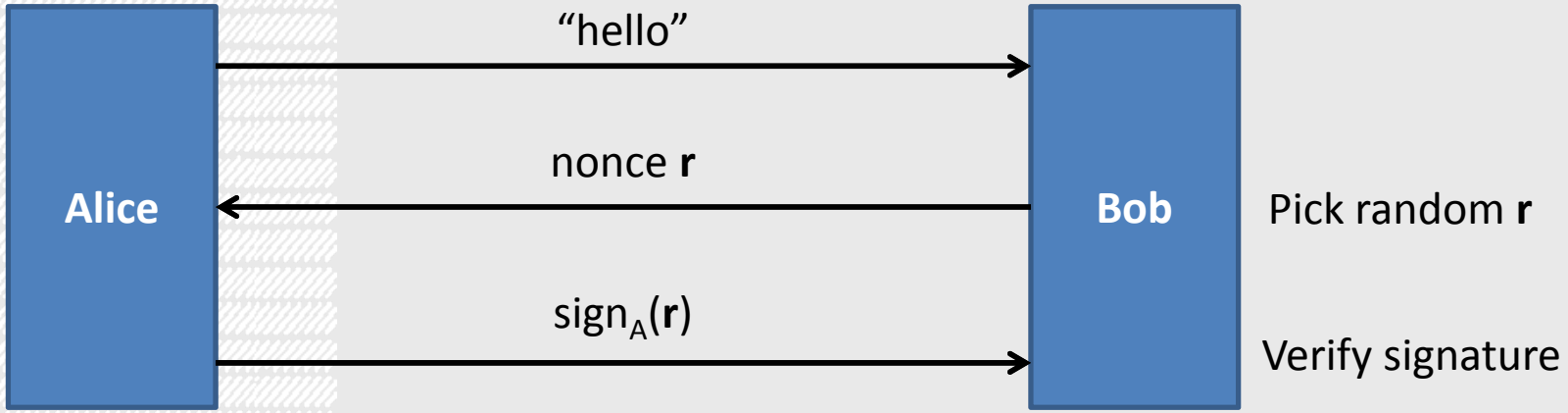
r' prevents a chosen plaintext attack (and as a challenge)

Both the user and server share the secret key **k**

Prevents eavesdropping

CHALLENGE-RESPONSE AUTHENTICATION USING ASYMMETRIC TECHNIQUES

Public key encryption / decryption used for digital signatures



No secrets stored on the server

Unlimited usage

Prevents eavesdropping

CHALLENGE-RESPONSE AUTHENTICATION USING 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 a series of challenge-response rounds

An adversary can cheat with very small probability

Whilst you can cheat on one round, the probability you'll succeed after **n** rounds decreases rapidly

ZERO KNOWLEDGE PROOFS

Problem:

Peggy wants to prove to Victor she knows some piece of information without revealing it

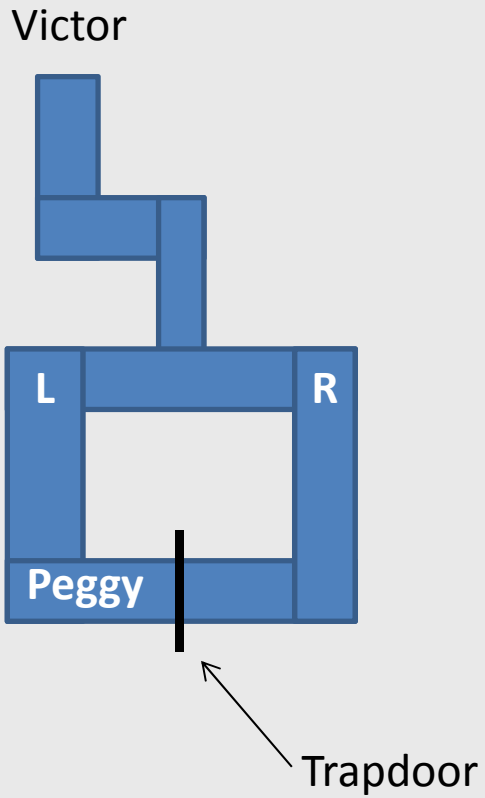
Proofs take the form of interactive protocols

- Victor asks Peggy a question
- If Peggy knows the answer she will always get it correct
- Otherwise, there is a small probability she can guess correctly
- Repeat asking questions until Victor is convinced

Already seen ZKPs have applications in authentication by challenge-response (e.g. proof of identity)

ALI BABA'S CAVE
QUISQUATER & GUILLOU (1989)

Peggy claims to know the password to open a trap door but doesn't want to tell it to Victor



Algorithm

1. Victor stands outside the cave
2. Peggy goes into random branch of the cave
3. Victor enters cave and tells Peggy to come out from the left or the right
4. If Peggy knows the password, can succeed every time
5. Repeat enough times until Victor is sure Peggy knows it

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
– Peggy cannot cheat Victor
– Victor cannot cheat Peggy
– Victor cannot pretend to be Peggy to any third party

ATTACKS ON ZKPS OF IDENTITY

The Mafia fraud Alice is eating at Fat Tony's Mafia Diner

- Fast Eddie is shopping at Bob's jewellery store
- Alice starts the ZKP identity protocol with Fat Tony
- Fat Tony radios Fast Eddie who starts a ZKP identity protocol with Bob
- Fat Tony and Fast Eddie as a communications channel
- Alice ends up being ripped off by the mafia

The Terrorist fraud Carlos the terrorist wants to enter the country

- Bob is scheming to help Carlos enter the country
- Carlos is challenged at the border by Alice with a ZKP of identity
- Carlos radios Bob and gets him to enter the ZKP identity protocol
- Alice thinks Carlos is Bob and lets him in

DINING CRYPTOGRAPHER'S PROBLEM

Problem:

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

- David Chaum (1988)

DINING CRYPTOGRAPHER'S PROBLEM

Algorithm:

1. Each cryptographer flips an unbiased coin (in secret)
2. Each shows the result to the person on the right
3. Each cryptographer states whether the two coins he can see are the same or different
4. If one of the cryptographers is the payer he says the opposite of what he sees
5. An odd number of differences means that a cryptographer has paid, otherwise the NSA paid
6. The algorithm is extensible to any number of diners

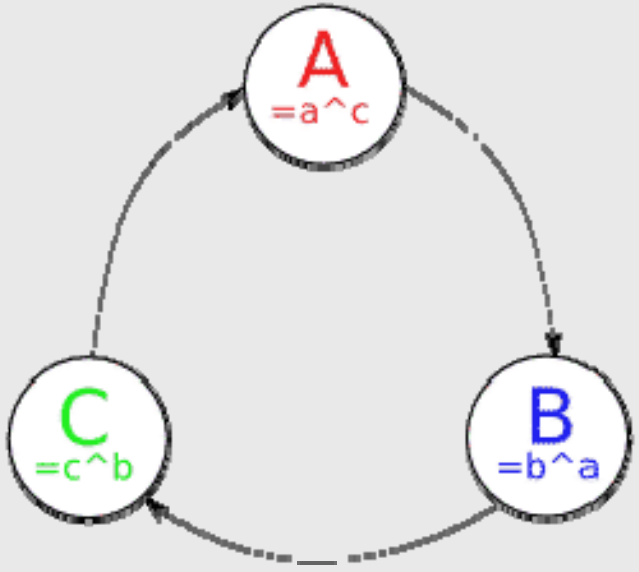
DINING CRYPTOGRAPHER'S PROBLEM

(no-one flips their value)

A ^ B ^ C
= (a ^ c) ^ (b ^ a) ^ (c ^ b)
= a ^ c ^ b ^ a ^ c ^ b
= c ^ b ^ c ^ b
= c ^ c
= 0

(one person flips their value)

A ^ B ^ C
= (a ^ c) ^ ¬ (b ^ a) ^ (c ^ b)
= (b ^ a) ^ ¬ (b ^ a)
= X ^ ¬X
= 1



DINING CRYPTOGRAPHER'S PROBLEM

Shows unconditional secrecy channels can be used to construct an unconditional sender (and receiver) untraceability channel.

Implies also that a public-key distribution system can be used to construct a secure sender-untraceability channel.

Otherwise known as anonymous broadcast

DINING CRYPTOGRAPHER'S PROBLEM

- It can also be extended to a full network (DC-net) by having the parties share a OTP rather than a coin-toss
  - Transfer many bytes at a time rather than a bit
    - XOR the OTPs between each party... the sender also XORs their message
- The biggest problem is that if two people try to transmit a message at the same time, both messages will be mangled
  - Overcome using a back off procedure similar to that in Ethernet CSMA/CD

CHALLENGE-RESPONSE USING ZERO KNOWLEDGE PROOFS

Say Alice knows x and wants to prove this to Bob without revealing any information about x.

Let G = <g> and somewhere is published $y=g^x$

Algorithm: (Discrete Log)

A → B: Alice chooses random r ϵ G and sends $z = yg^r = g^xg^r$ (*)

B → A: Bob tosses a coin e = {0,1} and sends to Alice

A → B: (**)

If e = 0, Alice sends $m = \log_g(z) = \log_g(g^xg^r) = x+r$

If e = 1, Alice sends $m = \log_g(zy^{-1}) = \log_g(g^xg^rg^{-x}) = r$

Bob verifies either

$g^m = z$ i.e. $g^{\log_g(z)} = g^{\log_g(g^xg^r)} = g^xg^r = z$

$g^m = zy^{-1}$ i.e. $g^{\log_g(zy^{-1})} = g^{\log_g(g^xg^rg^{-x})} = g^r = zy^{-1}$

CHALLENGE-RESPONSE USING ZERO KNOWLEDGE PROOFS

Eve can cheat the system if she knows the coin toss ahead of time

- If e = 0, she sends $z = g^r$ in the first stage (*) and r in the second (**)
- If e = 1, she sends $z = yg^r$ in the first stage (*), second doesn't matter (**)

As Eve can guess correctly the value of the coin toss half of the time on average, with probability 1/2 an imposter will suceed in breaking the protocol.

Doing the protocol **n** times reduces this probability of success to $\mathbf{0.5^n}$

(or you have a **1 in** $\mathbf{2^n}$ chance of being able to cheat)

NAIVE HTTP (WEB SERVER) AUTH

Basic Authentication

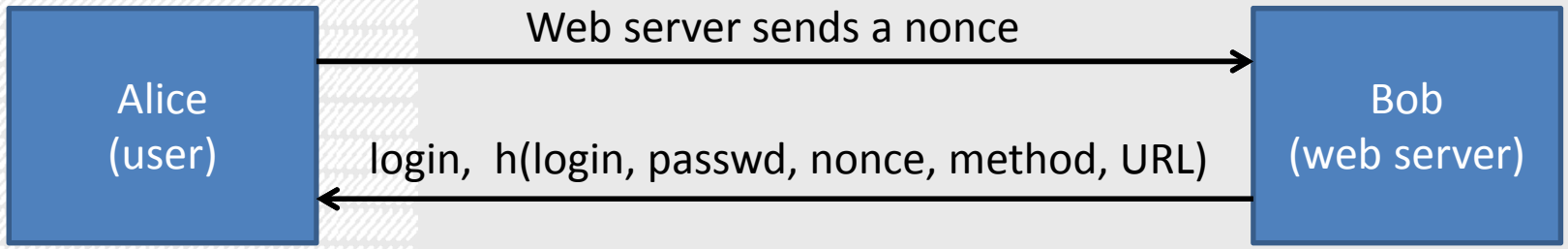
- access is segregated by realms
- simple base-64 encoding of username:password (no crypto)

WWW-Authenticate: Basic realm="Control Panel”

Authentication: Basic QWRtaW46Zm9vYmFy

Digest Authentication

- MD5 is used as the hash function



REFERENCES

Handbook of Applied Cryptography

- read § 10 – 10.4, §10.5
- skim § 10.4.1

NIST Electronic Authentication Guideline (December 2011)

- read *Appendix A: Estimating Entropy and Strength (Password Entropy)*