15-499:Algorithms and Applications

Cryptography V
- Kerberos
- Digital Cash

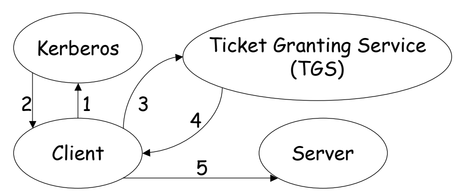
Cryptography Outline

Introduction: terminology and background
Primitives: one-way hash functions, trapdoors, …
Protocols: digital signatures, key exchange, ..
Number Theory: groups, fields, …
Private-Key Algorithms: Rijndael, DES, RC4
Cryptanalysis: Differential, Linear
Public-Key Algorithms: Knapsack, RSA, El-Gamal, Blum-Goldwasser
Case Studies: Kerberos, Digital Cash

Kerberos

A key-serving system based on Private-Keys (DES).
Assumptions
- Built on top of TCP/IP networks
- Many "**clients**" (typically users, but perhaps software)
- Many "**servers**" (e.g. file servers, compute servers, print servers, …)
- User machines and servers are potentially insecure without compromising the whole system
- A **kerberos server** must be secure.

Kerberos



1. Request *ticket-granting-ticket* (TGT)
2. <TGT>
3. Request *server-ticket* (ST)
4. <ST>
5. Request service

Kerberos V Message Formats

C = client S = server K = key
T = timestamp V = time range
TGS = Ticket Granting Service A = Net Address

Ticket Granting Ticket: $T_{C,TGS}$ = TGS,{C,A,V,$K_{C,TGS}$}K_{TGS}
Server Ticket: $T_{C,S}$ = S, {C,A,V,$K_{C,S}$ }K_S
Authenticator: $A_{C,S}$ = {C,T,[K]}$K_{C,S}$

1. Client to Kerberos: {C,TGS}K_C
2. Kerberos to Client: {$K_{C,TGS}$}K_C, $T_{C,TGS}$
3. Client to TGS: $A_{C,TGS}$, $T_{C,TGS}$
4. TGS to Client: {$K_{C,S}$}$K_{C,TGS}$, $T_{C,S}$
5. Client to Server: $A_{C,S}$, $T_{C,S}$

Possibly repeat

Kerberos Notes

All machines have to have synchronized clocks
- Must not be able to reuse authenticators

Servers should store all previous and valid tickets
- Help prevent replays

Client keys are typically a one-way hash of the password. Clients do not keep these keys.

Kerberos 5 uses CBC mode for encryption Kerberos 4 was insecure because it used a nonstandard mode.

Electronic Payments

Privacy
- Identified
- Anonymous

Involvement
- Offline (just buyer and seller)
  more practical for "micropayments"
- Online
  - Notational fund transfer (e.g. Visa, CyberCash)
  - Trusted 3rd party (e.g. FirstVirtual)

Today: "**Digital Cash**" (anonymous and possibly offline)

Some more protocols

1. Secret splitting (and sharing)
2. Bit commitment
3. Blind signatures

Secret Splitting

Take a secret (e.g. a bit-string **B**) and split it among multiple parties such that all parties have to cooperate to regenerate any part of the secret.

An implementation:

- Trent picks a random bit-string **R** of same length as **B**
- Sends Alice **R**
- Sends Bob **R xor B**

Generalizes to k parties by picking k-1 random values.

Secret Sharing

m out of n (m < n) parties can recreate the secret.

Also called an **(m,n)-threshold scheme**

An implementation (Shamir):

- Write secret as coefficients of a polynomial $GF(p^l)[x]$ of order m-1 ($n \leq p^l$).
  $p(x) = c_{m-1}x^{m-1} + \ldots + c\_1\ x + c\_0$
- Evaluate p(x) at n distinct points in $GF(p^l)$
- Give each party one of the results
- Any m results can be used to reconstruct the polynomial.

Bit Commitment

Alice commits a bit to Bob without revealing the bit (until Bob asks her to prove it later)

An implementation:

- **Commit**
  - Alice picks random r, and uses a one-way hash function to generate **y = f(r,b)**
    f(r,b) must be "unbiased" on b (y by itself tells you nothing about b).
  - Alice **sends** Bob **y**.
- **Open** (expose bit and prove it was commited)
  - Alice **sends** Bob **b** and **r**.

Example: y =Rijndael$_r$(000…b)

Blind Signatures

Sign a message **m** without knowing anything about **m**

Sounds dangerous, but can be used to give "value" to an anonymous message

- Each signature has meaning:
  $5 signature, $20 signature, …

Blind Signatures

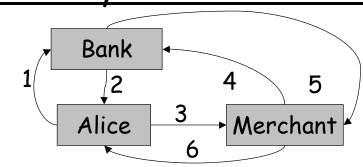
An implementation: based on RSA

Trent blindly signs a message m from Alice

- Trent has public key (e,n) and private key d
- Alice selects random $r < n$ and generates
  $m' = m\, r^e \bmod n$
  and sends it to Trent.
  This is called **blinding** m
- Trent signs it: $s(m') = (m\, r^e)^d \bmod n$
- Alice calculates:
  $s(m) = s(m')\, r^{-1} = m^d\, r^{ed-1} = m^d \bmod n$

Patented by Chaum in 1990.

An anonymous online scheme



1. Blinded Unique Random large ID (no collisions). Sig_{alice}(request for \$100).
2. $Sig_{bank\_\$100}$(blinded(ID)): signed by bank
3. $Sig_{bank\_\$100}$(ID)
4. $Sig_{bank\_\$100}$(ID)
5. OK from bank
6. OK from merchant

Minting: 1. and 2.
Spending: 3.-6.
Left out encryption

eCash

Uses the protocol

Bought assets and patents from Digicash
Founded by Chaum, went into Chapter 11 in 1998

Has not picked up as fast as hoped

- Credit card companies are putting up fight and transactions are becoming more efficient
- Government is afraid of abuse

Currently mostly used for Gift Certificates, but also used by Deutsche Bank in Europe.

The Perfect Crime

- Kidnapper takes hostage
- Ransom demand is a series of blinded coins
- Banks signs the coins to pay ransom
- Kidnapper tells bank to publish the coins in the newspaper (they're just strings)
- Only the kidnapper can unblind the coins (only he knows the blinding factor)
- Kidnapper can now use the coins and is completely anonymous

Chaum's protocol for offline anonymous cash

How do we prevent double payment without bank intervention?

Idea:

- If used properly, Alice stays anonymous
- If Alice spends a coin twice, she is revealed
- If Merchant remits twice, this is detected and Alice remains anonymous
- Must be secure against Alice and Merchant colluding
- Must be secure against one framing the other.

An amazing protocol

Chaum's protocol: **money orders**

u = Alice's account number (identifies her)

$r_0, r_1, \ldots, r_{n-1}$ = n random numbers

(ul_i, ur_i) = a secret split of u using r_i $(0 \leq i < n)$
e.g. using $(r_i, r_i \text{ xor } u)$

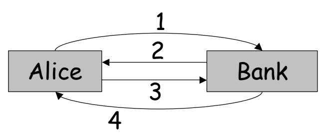
vl_i = a bit commitment of all bits of ul_i

vr_i = a bit commitment of all bits of ur_i

Money order:

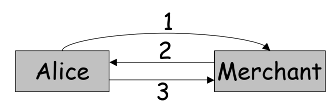
- Amount
- Unique ID
- $(vl_0,vr_0), (vl_1,vr_1), \ldots, (vl_{n-1},vr_{n-1})$

Chaum's protocol: **Minting**



1. Two blinded money orders and Alice's account #
2. A request to unblind and prove all bit commitments for one of the two orders (chosen at random)
3. The blinding factor and proof of commitment for that order
4. Assuming step 3. passes, the other blinded order signed

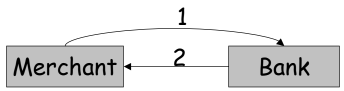
Chaum's protocol: **Spending**



1. The signed money order C (unblinded)
2. A random bit vector B of length n
3. For each i if $B_i = 0$ return bit values for ul_i else return bit values for ur_i
   Include all "proofs" that the ul or ur match vl or vr

Now the merchant checks that the money order is properly signed by the bank, and that the ul or ur match the vl or vr

Chaum's protocol: **Returning**



1. The signed money order
   The vector B along with the values of ul_i or ur_i that it received from Alice.
2. An OK, or fail

If fail, i.e., already returned:

1. If B matches previous order, the Merchant is guilty
2. Otherwise Alice is guilty and can be identified since for some i (where Bs don't match) the bank will have (ul_i, ur_i), which reveals her secret u (her identity).