Anonym communication with exercises

Holczer Tamás
4. 10. 2010.
Course: Security of Electronic Commerce
Outline – Anonym communication with exercises

- Summary of Tor and Chaum mixes
- Excercise: Generate a „Tor” network
- Numeric exercises
- Introduce Dining cryptographers for 3 participant
- Excercise: Generalize dining cryptographers
- Advanced techniques for anonym function evaluation
  - Brandt protocol
  - Anonym veto
  - Oblivious transfer
  - Yao’s garbled circuits
- Excercise: Generate a „Tor” network with active attackers (if enough time)
Summary of Chaum mixes and TOR

- Chaum mix
  - Chain of mixes
  - Onion encryption (layers of encryption)
  - Collect $\rightarrow$ decrypt $\rightarrow$ forward
  - No real time functionality

- Tor
  - Low latency
  - No waiting before forwarding
  - No protection against global adversary
Exercise: Manual TOR

- Task: build a „Tor” network manually
- Roles: user, webserver, intermediate node, attacker
- Attacker: global passive
- Steps:
  - Randomly select roles (remains hidden)
  - Define protocol
  - Key exchange (without attackers)
  - Random data distribution to be forwarded (without attackers), integer 0-99, random recipient
  - Data forwarding
  - Attackers’ guess
  - Discussion
Numeric exercises

- **Excercise 1.**
  - Linear network, N node
  - 2 user inserts a message with A time shift (t0=1, t1=A)
  - Each node sends to next hop with some delay
  - Distribution of delay is geometric (parameter $p$)
  - The two messages leave the network with C time interval B time later (t2=A+B, t3=A+B+C)
  - What is the probability that the messages leave the network in inverse order?

- **Excercise 2.**
  - Ring network, N node
  - Goal: sender anonymity
  - Each node sends to
    - right neighbor with probability $p$
    - left neighbor with probability $q$
    - destination D with probability $1-p-q$
  - What is the entropy of the attacker, when node A sends to D?
Solution

\[ E_1: \frac{(1-p)^C}{1+(1-p)^C} \]

\[ E_2: p_i = \left( p^i + q^{N-i} \right) \frac{1}{1-p^N} \frac{1}{1-q^N} \frac{1-(pq)^N}{1-pq} \]

\[ H = -\sum_{i=0}^{N-1} p_i \log p_i \]
**Dining cryptographers (Chaum)**

- Basic example of anonymous communication
- Offers sender and recipient anonymity
- Three cryptographers, who paid? Them or NSA?
- How to calculate private OR?

1. Everyone picks random $r_i$
2. Pass to right
3. Difference
4. Add message if any
5. Broadcast sum (xor)
6. Total sum = message

Everyone can be the recipient
No one knows the sender

$$S = s_1 + s_2 + s_3 = m_1 + m_2 + m_3$$
Exercise: Generalize DC

- Build DC in two separate groups
- Each group defines protocol
- Protocol description
- Key exchange with merged groups
- One node picked randomly from each group
- Run each protocol with merged groups
- Attack the other’s protocol
- Discussion
**DC: Generalization, problems, properties**

- N>3 nodes
- Every node must share a secret bit with every other node
  \[ S_i = \sum s_{i,j} + m_i \]
- Every node shares a key with the two neighbours (ring)
  - Two attacker can divide the anonymity set to two
- In general, attackers can divide the graph of users into smaller anonymity sets
- Collision (more than 1 sender) leads to ambiguous result
- Problems:
  - Key management, new users
  - Result can inverted maliciously
- Non-interactive, unconditional
Socialist millionaries’ protocol (Brandt) – Building blocks

- Similar to „Who is richer? My wealth is a secret!” (Yao 82)
- El Gamal Encryption (brief reminder):
  - \( p,q \) large primes
  - \( p-1=kq \) for some \( k \)
  - private key: \( x \in \mathbb{Z}_q \)
  - public key: \( y=g^x \)
  - encryption: \( (a,b) = (my^r, gr) \)
  - decryption: \( a/b^x=my^r/gr^x=my^r/y^r=m \)
- Homomorphism of El Gamal Encryption:
  - same key, product of two messages
  - \( (a_1a_2, b_1b_2) = (m_1m_2y^{r_1+r_2}, gr^{1+r_2}) \)
  - \( a_1a_2/b_1^xb_2^x = m_1m_2y^{r_1+r_2}/ g^{xr_1+xr_2} = m_1m_2y^{r_1+r_2}/y^{r_1+r_2}=m_1m_2 \)
- El Gamal encryption is semantical secure if DDH problem is intractable (DDH: knowing \( g^a,g^b \), hard to distinguish between \( g^{ab} \) and \( g^c \))
Socialist millionaries’ protocol (Brandt) – Building blocks

- **Distributed key generation:**
  - $x_i$ chosen at random by each participant
  - $y_i = g^{x_i}$ is broadcast with proof of knowledge of $x_i$ (later)
  - $y = \prod y_i$ is the public key
  - $x = \sum x_i$ is the private key

- **Distributed decryption:**
  - $(a, b)$ cyphertext
  - $b^{x_i}$ is broadcast with proof of equality of logarithm of $b^{x_i}$ and $y_i$
  - $m = a / \prod b^{x_i}$

- **Random exponentiation:**
  - $M_i$ random number
  - $(a^{M_i}, b^{M_i})$ is broadcast with proof of equality of logarithm of $a^{M_i}$ and $b^{M_i}$
  - $(a^M, b^M) = \prod (a^{M_i}, b^{M_i})$
Socialist millionaries’ protocol (Brandt) – Building blocks

- **Proof of knowledge of discrete logarithm (interactive form):**
  - Alice and Bob know \( v \) and \( g \), but only Alice knows \( x \), so that \( v = g^x \).
  - \( A \rightarrow B: a = g^z, \) \( z \) random value
  - \( B \rightarrow A: c \) random value
  - \( A \rightarrow B: r = (z+cx) \mod q \)
  - \( B \) checks: \( av^c = g^z (g^x)^c = g^{(z+cx)} = g^r \)

- **Proof of knowledge of discrete logarithm (non-interactive form):**
  - Alice and Bob know \( v \) and \( g \), but only Alice knows \( x \), so that \( v = g^x \).
  - \( e = H(g,r,v) \)
  - \( r = g^k \) \( s = k-xe \) proof: \( (r,s) \)
  - verification: \( g^s v^e = g^{k-xe} g^{xe} = g^k = r \)

- **Proof of equality of two discrete logarithms (interactive form):**
  - Alice and Bob know \( v,w, g1, \) and \( g2 \), but only Alice knows \( x \), so that \( v = g1^x \) and \( w = g2^x \)
  - \( A \rightarrow B: a = g1^z \) and \( b = g2^z, \) \( z \) random value
  - \( B \rightarrow A: c \) random value
  - \( A \rightarrow B: r = (z+cx) \mod q \)
  - \( B \) checks: \( av^c = g1^z (g1^x)^c = g1^{(z+cx)} = g1^r \) and \( aw^c = g2^z (g2^x)^c = g2^{(z+cx)} = g2^r \)
Socialist millionaries’ protocol (Brandt) – Building blocks

- Proof that an encrypted value is one out of two values:
  - \( m \in (1, z) \), without revealing \( M \)

1. If \( m = 1 \), Alice chooses \( r_1, d_1, w \) at random and sends \((\alpha, \beta), a_1 = g^{r_1 \beta^{d_1}}, b_1 = y^{r_1 \left(\frac{\alpha}{z}\right)^{d_1}} \) and \( a_2 = g^w, b_2 = y^w \) to Bob.
   
   If \( m = z \), Alice chooses \( r_2, d_2, w \) at random and sends \((\alpha, \beta), a_1 = g^w, b_1 = y^w, a_2 = g^{r_2 \beta^{d_2}}, \) and \( b_2 = y^{r_2 \alpha^{d_2}} \) to Bob.

2. Bob chooses a challenge \( c \) at random and sends it to Alice.

3. If \( m = 1 \), Alice sends \( d_1, d_2 = c - d_1 \mod q, r_1, \) and \( r_2 = w - rd_2 \mod q \) to Bob.
   
   If \( m = z \), Alice sends \( d_1 = c - d_2 \mod q, d_2, r_1 = w - rd_1 \mod q, \) and \( r_2 \) to Bob.

4. Bob checks that \( c = d_1 + d_2 \mod q, a_1 = g^{r_1 \beta^{d_1}}, b_1 = y^{r_1 \left(\frac{\alpha}{z}\right)^{d_1}}, a_2 = g^{r_2 \beta^{d_2}}, \) and \( b_2 = y^{r_2 \alpha^{d_2}} \).
Socialist millionaries’ protocol (Brandt) – The veto protocol

- **1. round: Distributed key generation**
- **2. round:** $E(b_i)$
  - $d_i = 1$ for agree
  - $d_i = Y$ for veto ($Y$ publicly known constant)
  - $D(E(d_i)) \in (1, Y)$, can be proven
  - compute $\prod_i E(d_i)$
- **3. round:** $\prod_i (E(d_i))^M$
  - compute $\prod_i (E(d_i))^M$ M random value, not known to anyone
- **4. round:** $d = D_{joint}(\prod_i (E(d_i))^M)$,
- No veto: $d = 1$
- Veto: $d$ random value
Anonym veto protocol (Hao, Zielinski)

1. round
- $x_i$ private key computed
- $g^{x_i}$ is broadcast (with knowledge proof of $x_i$)
- compute $g^{y_i}$:

$$g^{y_i} = \frac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^{N} g^{x_j}}$$

2. round
- broadcast (w kp):

$$g^{c_iy_i} = \begin{cases} g^{r_iy_i} & \text{if veto, } r_i \text{ random} \\ g^{x_iy_i} & \text{if no veto} \end{cases}$$

- compute:

$$D = \prod_{i=1}^{N} g^{c_iy_i}$$
Anonym veto protocol (Hao, Zielinski)

- If no veto: \( D=1 \)

\[
D = \prod_{i=1}^{N} g^{c_i y_i} = \prod_{i=1}^{N} g^{x_i y_i} = \prod_{i=1}^{N} \left( \prod_{j=i+1}^{N-1} g^{x_j} \right)^{x_i} = 
\]

\[
\prod_{i=1}^{N} \prod_{j=i+1}^{N} g^{x_j x_i} = \prod_{j=1}^{N} \prod_{i=j+1}^{N-1} g^{x_j x_i} = \prod_{j=1}^{N} \prod_{i=j+1}^{N-1} g^{x_j x_i} = i \leftrightarrow j \quad 1 
\]

- If veto: \( D=\text{random} \)
## Comparison

<table>
<thead>
<tr>
<th>Author</th>
<th>Year</th>
<th>Round</th>
<th>Security</th>
<th>Total traffic</th>
<th>Total comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaum</td>
<td>’88</td>
<td>2</td>
<td>unconditional</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Brandt</td>
<td>’05</td>
<td>4</td>
<td>DDH</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Hao Zielinski</td>
<td>’06</td>
<td>2</td>
<td>DDH</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Oblivious transfer with simple example

- Oblivious transfer: similar to PIR, but
  - the index of the record must remain unknown to the database (PIR)
  - other values in the database must remain unknown to the client
- RSA based k out of N oblivious transfer (OT$_k^N$) (~Digicash)

\[
\begin{align*}
X_0 = m_0^e, \ldots, X_N = m_N^e \\
Y_0 = X_{k_0} k_0^e, \ldots, Y_{k-1} = X_{k_{k-1}} k_{k-1}^e \\
C_0 = Y_0^d, \ldots, C_{k-1} = Y_{k-1}^d
\end{align*}
\]

Yao’s garbled circuit

- Secure two party computation
- Generalization of DC and veto
- Goal: compute $f(x,y)$
- Constraint: $x,y$ must be hidden from the other participant (some information can leak from $f(x,y)$)

Steps:
- $f$ is represented as combinatorial circuit
- The circuit is garbled
- Oblivious transfer
- The circuit is evaluated
Garbled gate with two input

- Alice generates four keys for inputs $x, y$: $K_{x=0}$, $K_{x=1}$, $K_{y=0}$, $K_{y=1}$
- Alice creates four boxes $B_{00} = f(0,0)$, $B_{01} = f(0,1)$, $B_{10}…$
- Each box is encrypted with the corresponding keys:
  \[ B'_{uv} = E(K_{x=u}||K_{y=v}, B_{uv}) \]
- Boxes are sent to Bob in random order
- Bob gets $K_{x=a}$ from Alice
- Bob uses OT to get $K_{y=b}$
- Bob decrypts $B'_{ab}$
- Analysis:
  - Alice does not know Bob’s input (OT)
  - Bob does not know Alice’s input (encrypted)
  - Active adversary? (results not discussed here)
- Larger circuit: output is key (instead of signal) for next gate
- Implementation: FairPlay compiler
Excercise: Manual TOR

- Task: build a „Tor” network manually
- Roles: user, webserver, intermediate node, attacker
- Attacker: global active
- Steps:
  - Randomly select roles (remains hidden)
  - Define protocol
  - Key exchange (without attackers)
  - Random data distribution to be forwarded (without attackers), integer 0-99, random recipient
  - Data forwarding
  - Attackers’ guess
  - Discussion