Intermediate Protocols
Part III

Lecture 3

Schneier: Ch 4
Building Block Protocols

- Secret Splitting
- Secret Sharing
- Timestamping Services
- Bit Commitment
- Fair Coin Flips
- Mental Poker
- Group Signatures
Fair Coin Flips

- **Problem:**
  - Bit is 0 Alice wins
  - Bob chooses b=1, sends to Alice
  - *Afterwards*, Alice chooses a=1
  - Alice can influence outcome of the final bit!
Fair Coin Flips: Original Solution

Manuel Bloom. Coin flipping by Telephone: A Protocol for Solving Impossible Problems
Fair Coin Flips: Properties

1. Alice must flip coin before Bob guesses
2. Alice must be unable to flip again her coin after hearing Bob’s guess
3. Bob must be unable to know a before making his guess
Fair Coin Flips: with Hashes

- Choose value \( a \)
- \( C = H(a) \)
- \( a \) is even/\( a \) is odd
- Guess: is \( a \) even or odd?
- If guess is correct, Bob wins (coin is head)

- Security rests on the hash function
Fair Coin Flips: with Public Key Crypto

- Need commutative cryptosystems
- \( D_{pr1}(E_{pk2}(E_{pk1}(M))) = E_{pk2}(M) \)
- Works for some public key cryptosystems
  - RSA with identical moduli
Fair Coin Flips: with Public Key Crypto

1. Alice generates `pk_A`, `pr_A`.
2. `E_A(M_0), E_A(M_1)`.
3. `E_B(E_A(M_b))`.
4. `D_A(E_B(E_A(M_b))) = E_B(M_b)`.
5. `M_b = “coin”, R_b`.
6. `D_B(E_B(M_b)) = M_b`.
7. A verifies that “coin” corresponds to `R_b`.

Flip coin `b` (0 or 1).

Generate `pk_B`, `pr_B`.

Generate `M_0 = “heads”, R_0`.

Generate `M_1 = “tails”, R_1`. 
Building Block Protocols

- Secret Splitting
- Secret Sharing
- Timestamping Services
- Bit Commitment
- Fair Coin Flips
- Mental Poker
- Group Signatures
Mental Poker: Two Players

- Generalization of fair coin flip

Alice
- Generate \( \text{pk}_A, \text{pr}_A \)
- Decrypt messages
- Decrypt to get her “hand”

Bob
- Generate \( \text{pk}_B, \text{pr}_B \)
- Pick 5 at random
- Decrypt to get his “hand”
- Pick 5 other at random

1. \( E_A(M_1), .. E_A(M_{52}) \)
2. \( E_B(E_A(M_1)), ..., E_B(E_A(M_5)) \)
3. \( E_B(M_1), ..., E_B(M_5) \)
4. \( E_B(M_1), ..., E_B(M_5) \)
5. \( E_B(M_6), ..., E_A(M_{10}) \)
Mental Poker: Two Players (cont’d)

- More cards can be dealt later
  - Using the same procedure
- At the end of the game Alice and Bob
  - Reveal their cards
  - And private keys
  - *For verification purposes*
- A and B’s public keys are not public !!!
  - Why ?
Mental Poker: Three Players

1. Alice generates $pk_A, pr_A$.
2. Alice generates 52 messages $M_n$, $n = 1..52$.

3. Alice encrypts $M_n$ with her key: $E_A(M_n)$.
4. Alice encrypts $E_A(M_n)$ with Bob's key: $E_B(E_A(M_n)) : 5$ of them.
5. Alice encrypts $E_A(M_n)$ with her key: $E_C(E_A(M_n)) : 5$ of them.
6. Alice encrypts $E_A(M_n)$ with her key: $E_C(M_n)$.
7. Alice encrypts $M_n$ with Bob's key: $E_B(M_n)$.

8. Carol encrypts $E_A(M_n)$ with her key: $E_C(E_A(M_n)) : 5$ of them.
9. Carol encrypts $M_n$ with her key: $E_C(M_n)$.
10. Carol encrypts $E_A(M_n)$ with her key: $E_A(M_n) : 5$ of them.

Pick 5 at random.

Bob generates $pk_B, pr_B$.

Pick 5 at random.

Carol generates $pk_C, pr_C$.

Pick 5 at random (of 47).

Pick 5 at random (of 42).
Mental Poker: Three Players (cont’d)

- If B (or C) want a card
  - Whoever has the deck sends them a card
  - B (or C) encrypts and sends the card to A for decryption
- If A wants a card
  - Whoever has the deck sends A a card
  - A can decrypt
Three Players: Verification

- If Alice wins
  - Reveal her hand and keys
  - Bob verifies that step 2 was correct
  - Carol verifies that step 10 was correct
- If Bob (or Carol) wins
  - Reveal the hand and keys
  - Alice verifies the winner’s encryption in step 3 or 6
Three Players: Attack

- Alice can cheat
  - By colluding with one other
  - Against the third player
  - How?
Anonymous Key Distribution

- Users need keys (e.g., public/private key pairs)
  - Validly created
    - By a trusted party
  - Anonymous
    - Private party does not know the key
- How?
  - Use mental poker
Anonymous Key Distribution (cont’d)

Alice

Generate \( pk_A, pr_A \)
Both are secret

Trent T

Generate \( pk_T, pr_T \)
Both are secret

1. **Continuously generate keys:** \( k_1, \ldots, k_n \)

2. \( E(pk_T k_1), \ldots, E(pk_T k_n) \)

3. **Pick one at random**

   *Wait a while (Why?)*

4. \( E_A(E(pk_T k_i)) \)

5. \( D(pr_T E_A(E(pk_T k_i))) \)

6. \( E_A(k_i) \)

7. \( D_A(E_A(k_i)) = k_i \)
Building Block Protocols

- Secret Splitting
- Secret Sharing
- Timestamping Services
- Bit Commitment
- Fair Coin Flips
- Mental Poker
- Group Signatures
Digital Signatures

- Why do we need digital signatures?
  - Verify author of message
  - Verify time of message
  - Authenticate message contents

- Verifiable by third parties
  - To resolve disputes
Digital Signature Model

Plaintext → Signature Algorithm → Signature → Verification Algorithm → Valid! / Invalid!

Private Key → Public Key

Plaintext
In Real Life

Alice

Bob

Malory

pubKey_B - public

privKey_B - private

Verify(M, S, pubKey_B) = true!

M' = I owe Malory $1000 for same S

Bob's Signature

M' = I owe Malory $1000 for new S'

Intercept

Malory

S = Sign(M, privKey_B)

M = I owe Alice $1000
In Real Life

Alice

Bob

Verify\((M, S, \text{pubKey}_B)\) = true!

Bob Cannot Deny Signature S!

M = I owe Alice $1000

S = \text{Sign}(M, \text{privKey}_B)
Group Signatures

- How to prove you are part of a group
  - Authentication
  - Privately
  - In case of misbehavior, reveal identity

1. Only group members can sign messages
2. Receiver of signature can verify validity
3. Receiver of signature cannot determine identity
4. In case of dispute, signature can be opened to identify culprit
Group Signatures: Trusted Arbitrator

1. Generate \((pb^A_1, pr^A_1), \ldots, (pb^A_m, pr^A_m)\)

2. \(pr^A_1, \ldots, pr^A_m\)

3. To sign: Pick random \(pr^A_i\)

4. \(S^A_i(M)\)

5. To verify: Use \(pb^A_i\) (public)

- N participants
- Publish all public keys
  - \(N \times m\) keys

In case of dispute: 
T knows mapping \([pb^A_i, A]\)