Introduction to Cryptography
Lecture 13

Benny Pinkas

Electronic cash

Simple electronic checks

- A payment protocol:
  - Sign a document transferring money from your account to another account
  - This document goes to your bank
  - The bank verifies that this is not a copy of a previous check
  - The bank checks your balance
  - The bank transfers the sum

- Problems:
  - Requires online access to the bank (to prevent reusage)
  - Expensive.
  - The transaction is traceable (namely, the bank knows about the transaction between you and Alice).

First try at a payment protocol

- Withdrawal
  - User gets bank signature on {I am a $100 bill, #1234}
  - Bank deducts $100 from user's account
- Payment
  - User gives the signature to a merchant
  - Merchant verifies the signature, and checks online with the bank to verify that this is the first time that it is used.
- Problems:
  - As before, online access to the bank, and lack of anonymity.
- Advantage:
  - The bank doesn't have to check online whether there is money in the user's account.
  - In fact, there is no real need for the signature, since the bank checks its own signature.

Anonymous cash via blind signatures

- In order to preserve payer's anonymity the bank signs the bill without seeing it
  - (e.g. like signing on a carbon paper)

- RSA Blind signatures (Chaum)
- RSA signature: $(H(m))^{1/e}$ mod n
- Blind RSA signature:
  - Alice sends Bob ($r^e H(m)$) mod n, where r is a random value.
  - Bob computes $(r^e H(m))^{1/e} = r H(m)^{1/e}$ mod n, and sends to Alice.
  - Alice divides by r and computes $H(m)^{1/e}$ mod n

- Problem: Alice can get Bob to sign anything, Bob does not know what he is signing.

Enabling the bank to verify the signed value

- “cut and choose” protocol
- Suppose Alice wants to sign a \$20 bill.
  - A \$20 bill is defined as *H(random index,\$20)*.
  - Alice sends to bank 100 different \$20 bills for blind signature.
  - The bank chooses 99 of these and asks Alice to unblind them (divide by the corresponding *r* values). It verifies that they are all \$20 bills.
  - The bank blindly signs the remaining bill and gives it to Alice.
  - Alice can use the bill without being identified by the bank.

- If Alice tries to cheat she is caught with probability 99/100.
- 100 can be replaced by any parameter *m*.
- But we would like to have an exponentially small cheating probability.

Exponentially small cheating probability

- Define that a \$20 bill in a new way:
  - The bill is valid if it is the RSA signature of the multiplication of 50 values of the form *H(x),* (where *x="random index,\$20"*).

- The withdrawal protocol:
  - Alice sends to the Bank $z_1, z_2, \ldots, z_{100}$ (where $z_i = r_i^{\,e} \cdot H(x_i)$).
  - The Bank asks Alice to reveal ½ of the values $z_i = r_i^{\,e} \cdot H(x_i)$.
  - The Bank verifies them and extracts the e^{th} root of the multiplication of all the other 50 values. Alice divides the results by the multiplication of the corresponding r_i values.
- Payment: Alice sends the signed bill and reveals the 50 preimage values. The merchant sends them to the bank which verifies that they haven't been used before.

- Alice can only cheat if she guesses the 50 locations in which she will be asked to unblind the z_is, which happens with probability $\sim 2^{-100}$.

Online vs. offline digital cash

- We solved the anonymity problem, while verifying that Alice can only get signatures on bills of the right value.

- The bills can still be duplicated
- Merchants must check with the bank whenever they get a new bill, to verify that it wasn't used before.

- A new idea:
  - During the payment protocol the user is forced to encode a random identity string (RIS) into the bill
  - If the bill is used twice, the RIS reveals the user's identity and she loses her anonymity.

Offline digital cash

Withdrawal protocol:

- Alice prepares 100 bills of the form
  - {I am a $20 bill, *#1234*, $y_1,y'_1,y_2,y'_2,\ldots,y_m,y'_m$}
  - S.t. $\forall i\ y_i=H(x_i)$, $y'_i=H(x'_i)$, and it holds that $x_i \oplus x'_i$ *=Alice's id,* where *H()* is a collision resistant function.

- Alice blinds these bills and sends to the bank.

- The bank asks her to unblind 99 bills and show their x_i,x'_i values, and checks their validity.
  - (Alternatively, as in the previous example, Alice can do a check with fails with only an exponential probability.)

- The bank signs the remaining blinded bill.

Offline digital cash

Payment protocol:

- Alice gives a signed bill to the vendor
  - {I am a $20 bill, #*1234*, $y_1, y'_1, y_2, y'_2, \ldots, y_m, y'_m$}
- The vendor verifies the signature, and if it is valid sends to Alice a random bit string $b = b_1 b_2 \ldots b_m$ of length *m.*
- $\forall i$ if $b_i = 0$ *Alice returns* x_i, otherwise $(b_i = 1)$ she returns x'_I
- The vendor checks that $y_i = H(x_i)$ or $y'_i = H(x'_i)$ (depending on b_i*).* If this check is successful it accepts the bill. (Note that Alice's identity is kept secret.)

- Note that the merchant does not need to contact the bank during the payment protocol.

Offline digital cash

- The merchant must deposit the bill in the bank. It cannot use the bill to pay someone else.
  - Because it cannot answer challenges b^* different than the challenge b it sent to Alice.

- How can the bank detect double spenders?
  - Suppose two merchants M and M^* receive the same bill
  - With very high probability, they ask Alice *different* queries b, b^*
  - There is an index i for which $b_i=0$, $b^*_i=1$. Therefore M receives x_i and M^* receives x'_i.
  - When they deposit the bills, the bank receives x_i and x^*_i, and can compute $x_i \oplus x'_i = Alice's\ id.$

Secure multi-party computation

- Problem statement:
  - n players P_1, P_2,…, P_n
  - Player P_i has input x_i
  - There is a known function $f(x_1,\ldots,x_n)= (y_1,\ldots y_n)$
- Goals:
  - P_i should learn y_i, and nothing else (except for what can be computed from x_i and y_i)
  - This property should also hold for coalitions of corrupt parties (e.g., $P_1,\ldots,P_{n/3}$ should learn nothing but $x_1,\ldots,x_{n/3},y_1,\ldots,y_{n/3}$)
  - Security should hold even against malicious parties
- Examples…

More on MPC

- Generality: MPC is extremely general, covers almost all protocol problems.
- We will define a protocol, which tells each party which messages to send to other parties.
- Adversaries:
  - Semi-honest vs. malicious
    - Semi-honest (“honest but curious”) follow the protocol but try to deduce information from it
    - Malicious adversaries can behave arbitrarily
  - Static (decide in advance which parties to corrupt) vs. adaptive (decide on the fly which parties to corrupt)
  - Unbounded vs. probabilistic polynomial-time

Defining security

- It is not sufficient to list the desired properties that the protocol should satisfy
  - How can we be sure that we covered all properties?
- Basic security definition: comparison to an ideal scenario
  - In the ideal scenario there is a trusted party which receives $x_1,\ldots,x_n$, computes the function and sends y_i to P_i.
  - The real protocol is secure if its execution reveals no more than in the ideal scenario.
- The actual definition is much more complicated, in particular if we consider multiple invocations of the same protocol.

What is known

- Information theoretic scenario:
  - Semi-honest, adaptive adversary: any function can be computed iff adversary controls up to $t<n/2$ parties.
  - Malicious, adaptive adversary: any function can be computed iff adversary controls up to $t<n/3$ parties.
    - If broadcast is available, can withstand up to $t<n/2$.
- Cryptographic scenario:
  - Semi-honest, adaptive, polynomial-time adversary: assuming one-way trapdoor permutations exist, any function can be computed if $t<n$.
  - Malicious, adaptive, polynomial-time adversary: assuming one-way trapdoor permutations exist, any function can be computed if $t<n/2$.

An MPC protocol for semi-honest parties

- We will show a construction in the unconditional security scenario, against semi-honest, adaptive adversaries which control up to $t<n/2$ parties.

- The basic idea:
  - Any input value can be shared between the n participants, such that no t of them can reconstruct it.
  - It is possible to make computations on shared values.

- Initial step:
  - Write the function as an arithmetic circuit modulo a prime number p.

Arithmetic circuits

- Circuits where
  - Wires transfer values defined over a field
  - Gates implement + and *

- Note that arithmetic circuits can be much more compact than combinatorial (Boolean) circuits (with AND and OR gates). For example, for computing $a+b$ or $a \cdot b$.

- Any Boolean circuit can be implemented as a arithmetic circuit
  - True is represented as 1, false as 0.
  - AND(x,y) is implemented as x*y
  - OR(x,y) is implemented as x+y-x*y
  - NOT(x) is implemented as 1-x

t-out-of-*n* secret sharing

- Shamir's secret sharing scheme:
  - Choose a large prime and work in the field *Zp*.
  - The secret S is an element in the field.
  - Define a polynomial *P* of degree *t-1* by choosing random coefficients $a_1,\ldots,a_{t-1}$ and defining
    $P(x) = a_{t-1}x^{t-1}+\ldots+a_1x+\underline{S}$.
  - The share of party *j* is *(j, P(j)).*

An MPC protocol for n semi-honest parties, secure against t<n/2 parties.

- Each party P_i has an input x_i.
- The first step of the protocol:
  - Each P_i generates a (t+1)-out-of-n sharing of its input x_i
    - Namely, chooses a random polynomial $f_i()$ over Z_p^* such that $f_i(0)=x_i$.
    - Any subset of t shares does not leak any information about x_i
    - t+1 shares enable to reconstruct x_i using polynomial interpolation
  - Every P_i sends to each P_j ($j\neq i$) the value $f_i(j)$
- The protocol continues by induction from the input wires to the output wires.
  - We will show that for every gate, if the parties know shares of the input values, they can compute shares of the output values.

Computation stage

- All parties participate in the computation of every gate
- Addition gate: c= a+b
  - The parties must generate a sharing of c.
  - Namely, there should be a polynomial $f_c()$ of degree t, such that $f_c()$ is random except for $f_c(0)=c$
  - (Note that defining $f_c(x)=f_a(x)+f_b(x)$ will be fine)
  - Each P_i must receive the share $c_i=f_c(i)$
- The protocol:
  - Each player P_i already has shares of a and b.
  - Namely, P_i has shares $a_i=f_a(i)$ and $b_i=f_b(i)$ of polynomials $f_a()$ and $f_b()$ of degree t, for which $f_a(0)=a$ and $f_b(0)=b$.
  - P_i sets $c_i=a_i+b_i = f_a(i)+f_b(i) = f_c(i)$
  - No communication is needed for this computation.

Output phase

- Easier to describe than the protocol for multiplication gates
- Output wires
  - If output wire y_i must be learned by P_i, then all parties send it their shares of y_i.
  - P_i reconstructs the secret and learns the output value.

Computation stage: multiplication gate

- Each player P_i already has shares $a_i=f_a(i)$ and $b_i=f_b(i)$.
- Needs to have a share d_i of $d=a\cdot b$.
- First attempt:
  - P_i sets $d_i=a_i\cdot b_i = f_d(i)$.
  - Obtains a share of $f_a() \cdot f_b()$
  - Indeed, $f_d(0) = d = a\cdot b$.
  - But $f_d()$ is of degree $2t$ and not t.
    - If we do this twice, the degree becomes $4t>n$ and n parties will not be able to reconstruct the secret.

Computing multiplication gates

- P_i sets $d_i = a_i \cdot b_i = f_d(i)$.
- $f_d(i)$ is of degree $2t < n$.
- We know the values of (Lagrange) coefficients $r_1,..,r_n$ such that $d = f_d(0) = a \cdot b = r_1 f_d(1) + \ldots + r_n f_d(n) = r_1 d_1 + \ldots + r_n d_n$.

- Each P_i creates a random polynomial g_i of degree t such that $g_i(0) = d_i$.
- Consider $G(x) = \sum_{i=1}^{n} r_i \cdot g_i(x)$
  - This a polynomial of degree t.
  - $G(0) = \sum_{i=1}^{n} r_i \cdot g_i(0) = \sum_{i=1}^{n} r_i \cdot d_i = d$.
- Now, if only we could provide each P_j with $G(j) = \sum_{i=1}^{n} r_i \cdot g_i(j)$ …

Computing multiplication gates

- P_i sends to every P_j the value $g_i(j)$

- Every P_j receives $g_1(j),\ldots,g_n(j)$, and computes $G_j = \sum_{i=1}^{n} r_i \cdot g_i(j) = G(j)$

- This is the desired share of $a \cdot b$:
  - it is a value of the polynomial $G(x)=\sum_{i=1}^{n} r_i \cdot g_i(x)$,
  - of degree t,
  - for which $G(0)= a \cdot b$.

Computing the entire circuit

- The parties do this computation for every gate
- Opening the outputs
  - At the end of the circuit, for each output y_j which should be known to P_j, it holds that the parties hold shares of a polynomial $f(x)$ of degree t such that $f(0)=y_j$.
- Each party P_i sends $f(i)$ to P_j.
- P_j interpolates $f(0)=y_j$.

Properties

- Correctness: straightforward
- Privacy: For every set of t players, it holds that all values they see in the protocol are shares of (t+1)-out-of-n secret sharing schemes.
  – Therefore all their t shares are uniformly distributed.
  – The proof needs to make sure that this property holds even if adversary gets shares of a,b, and a·b
- Overhead:
  – $O(n^2)$ messages for every multiplication gate.
  – Number of communication rounds is linear in the depth of the circuit (where only multiplication gates are counted).