CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 24
April 23, 2013

Privacy-preserving Boolean Function Evaluation

Boolean functions computed by circuits

Let $\bar{z} = f(\bar{x}, \bar{y})$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are bit strings of lengths n_x, n_y, and n_z, respectively, and f is a Boolean function computed by a polynomial size Boolean circuit C_f with $n_x + n_y$ input wires and n_z output wires.

In a *private evaluation* of C_f, Alice furnishes the (private) input data to the first n_x input wires, and Bob furnishes the (private) input data for the remaining n_y input wires. The n_z output wires should contain the result $\bar{z} = f(\bar{x}, \bar{y})$. The corresponding functionality is

$$F(\bar{x}, \bar{y}) = (\bar{z}, \bar{z}).$$

Alice and Bob should learn nothing about each other's inputs or the intermediate values of the circuit, other than what is implied by their own inputs and the output values $\bar{z}$.

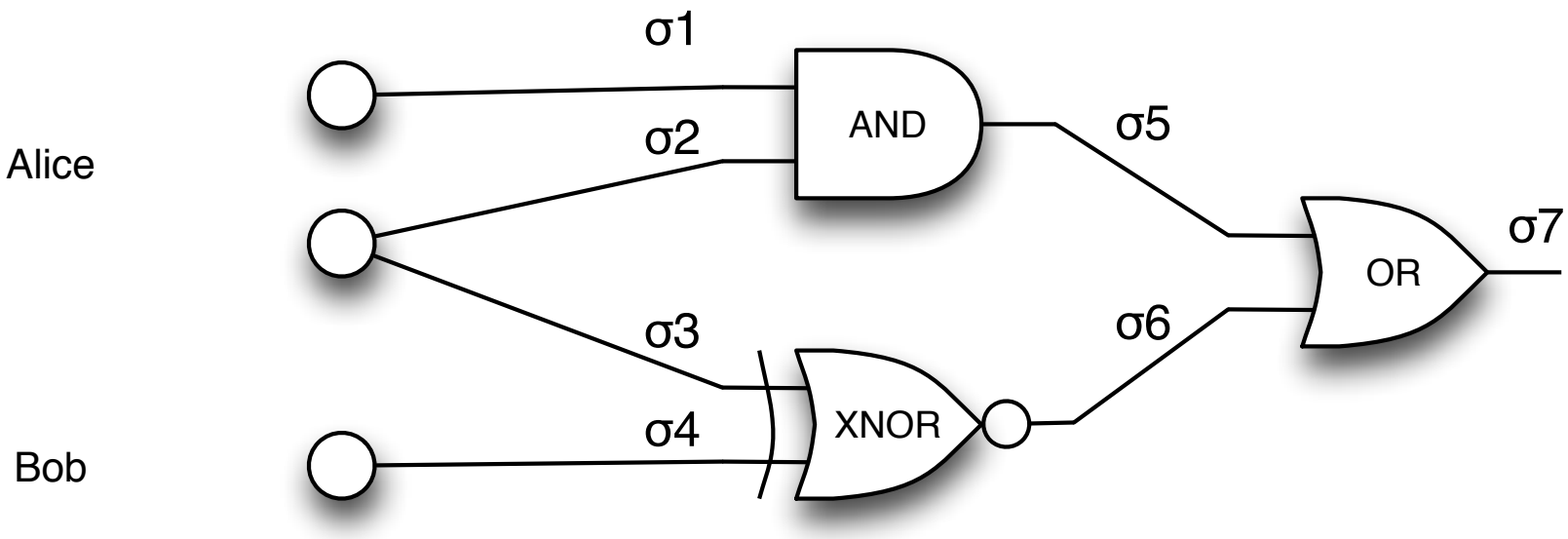
Circuit evaluation

An evaluation of a circuit assigns a Boolean value σ_w to each wire of the circuit. The input wires are assigned the corresponding input values.

Let G be a gate with input wires u and v and output wire w that computes the Boolean function $g(x, y)$. In a correct assignment, $\sigma_w = g(\sigma_u, \sigma_v)$.

A complete evaluation of the circuit first assigns values to the input wires and then works its way down the circuit, assigning a value to the output wire of any gate whose inputs have already received values.

A Boolean circuit



Private circuit evaluation

In a *private circuit evaluation*,

- Both Alice and Bob learn the output values of the circuit;
- Neither Alice nor Bob gain any information about each other's private input values except for whatever is implied by their own input values and the circuit output.

We present two different schemes for privately evaluating a circuit:

- Value shares;
- Garbled circuits.

Circuit Evaluation Using Value Shares

Value shares

In a private evaluation using *value shares*, we split each value σ_w into two random shares a_w and b_w such that $\sigma_w = a_w \oplus b_w$.

- Alice knows a_w; Bob knows b_w.
- Neither share alone gives any information about σ_w, but together they allow σ_w to be computed.

After all shares have been computed for all wires, Alice and Bob exchange their shares a_w and b_w for each output wire w.

They are both then able to compute the circuit output.

Obtaining the shares

We now describe how Alice and Bob obtain their shares while maintaining the desired privacy.

There are three cases, depending on whether w is

1. An input wire controlled by Alice;
2. An input wire controlled by Bob;
3. The output wire of a gate G.

Alice's input wires

1. Input wire controlled by Alice:

   Alice knows σ_w.

   She generates a random share $a_w \in \{0, 1\}$ for herself and sends Bob his share $b_w = a_w \oplus \sigma_w$.

Bob's input wires

2. Input wire controlled by Bob:

   Bob knows σ_w.

   Alice chooses a random share $a_w \in \{0, 1\}$ for herself.

   She prepares a table T:

σ	$T[\sigma]$
0	a_w
1	$a_w \oplus 1$.

   Bob requests $T[\sigma_w]$ from Alice via OT_1^2 and takes his share to be $b_w = T[\sigma_w] = a_w \oplus \sigma_w$.

Obtaining shares for gate output wires

3. Output wire of a gate G:

   Let G have input wires u, v and compute function $g(x, y)$.
   Alice chooses random share $a_w \in \{0, 1\}$ for herself.
   She computes the table

$$\begin{aligned} T[0,0] &= a_w \oplus g(a_u, a_v) \\ T[0,1] &= a_w \oplus g(a_u, a_v \oplus 1) \\ T[1,0] &= a_w \oplus g(a_u \oplus 1, a_v) \\ T[1,1] &= a_w \oplus g(a_u \oplus 1, a_v \oplus 1) \end{aligned}$$

   (Equivalently, $T[r, s] = a_w \oplus g(a_u \oplus r, a_v \oplus s)$.)

   Bob requests $T[b_u, b_v]$ from Alice via OT_1^4 and takes his share to be $b_w = T[b_u, b_v] = a_w \oplus g(\sigma_u, \sigma_v)$.

Remarks

1. Alice and Bob's shares for w are both independent of σ_w.
   - Alice's share is chosen uniformly at random.
   - Bob's share is always the XOR of Alice's random bit a_w with something independent of a_w.
2. This protocol requires n_y executions of OT_1^2 to distribute the shares for Bob's inputs, and one OT_1^4 for each gate.[1]
3. This protocol assumes semi-honest parties.
4. This protocol generalizes readily from 2 to m parties.
5. Bob does not even need to know what function each gate G computes. He only uses his private inputs or shares to request the right line of the table in each of the several OT protocols.

[1] Note: The n_y executions of OT_1^2 can be eliminated by having Bob produce the shares for his input wires just as Alice does for hers. Our approach has the advantage of being more uniform since Alice is in charge of distributing the shares for all wires.

Circuit Evaluation Using Garbled Circuits

Garbled circuits

A very different approach to private circuit evaluation is the use of *garbled circuits*.

The idea here is that Alice prepares a garbled circuit in which each wire has associated with it a tag corresponding to 0 and a tag corresponding to 1.

Associated with each gate is a template that allows the tag that represent the correct output value to be computed from the tags representing the input values.

This is all done in a way that keeps hidden the actual values that the tags represent.

A sketch of the protocol

After creating the circuit, Alice, who knows all of the tags, uses OT_1^2 to send Bob the tags corresponding to values on the input wires that he controls.

She also sends him the tags corresponding to the values on the input wires that she controls.

Bob then evaluates the circuit all by himself, computing the output tag for each gate from the tags on the input wires.

At the end, he knows the tags corresponding to the output wires.

Alice knows which Boolean values those tags represent, which she sends to Bob (either before or after he has evaluated the circuit).

In this way, Bob learns the output of the circuit, which he then sends to Alice.

Role of the tags

The scrambled gate is a 4-line table giving the output tag corresponding to each of the possible 4 input values.

Each line of the table is encrypted differently.

The input tags to the gate allow the corresponding table item to be decrypted.

Evaluating the circuit then amounts to decrypting ones way though the circuit, gate by gate, until getting the output tag.

Remarks

1. The OT_1^2 protocol steps used to distribute the tags for the wires that Bob controls keeps his inputs private from Alice. The privacy of Alice's inputs and intermediate circuit values from Bob relies on the encryption function used to hide the association between tags and values.
2. The security of the protocol relies on properties of the encryption function that we have not stated.
3. This protocol requires only n_y executions of OT_1^2 and hence should be considerably faster to implement than the share-based protocol.
4. This protocol also assumes semi-honest parties.
5. Doesn't easily generalize to more than two parties.
6. Bob doesn't need to know the function each gate computes. He only needs the associated templates.

Bitcoins

Bitcoins

Bitcoins are a kind of digital cash. Some of their properties are:

- They are annonymous.
- Their supply is limited.
- There is cryptographic protection against double-spending and forgery.

Who uses them?

Bitcoins permit anonymous transactions.

They are apparently being used now for illicit transactions on internet web sites.

They are also being purchased by speculators who believe their value will go up.

Some people are betting that merchants will begin accepting payments in Bitcoins and that their value will rise as they gain acceptance.

How do they work?

- They live in a "database" that is shared among a large group of miners.
- Miners check the validity of transactions and then attempt to commit them to the master database.
- The validated database is broadcast to the miners in order to "commit" it.
- If enough miners accept the database as being the most recent, then it will (with high probability) always be accepted by a majority of the miners.

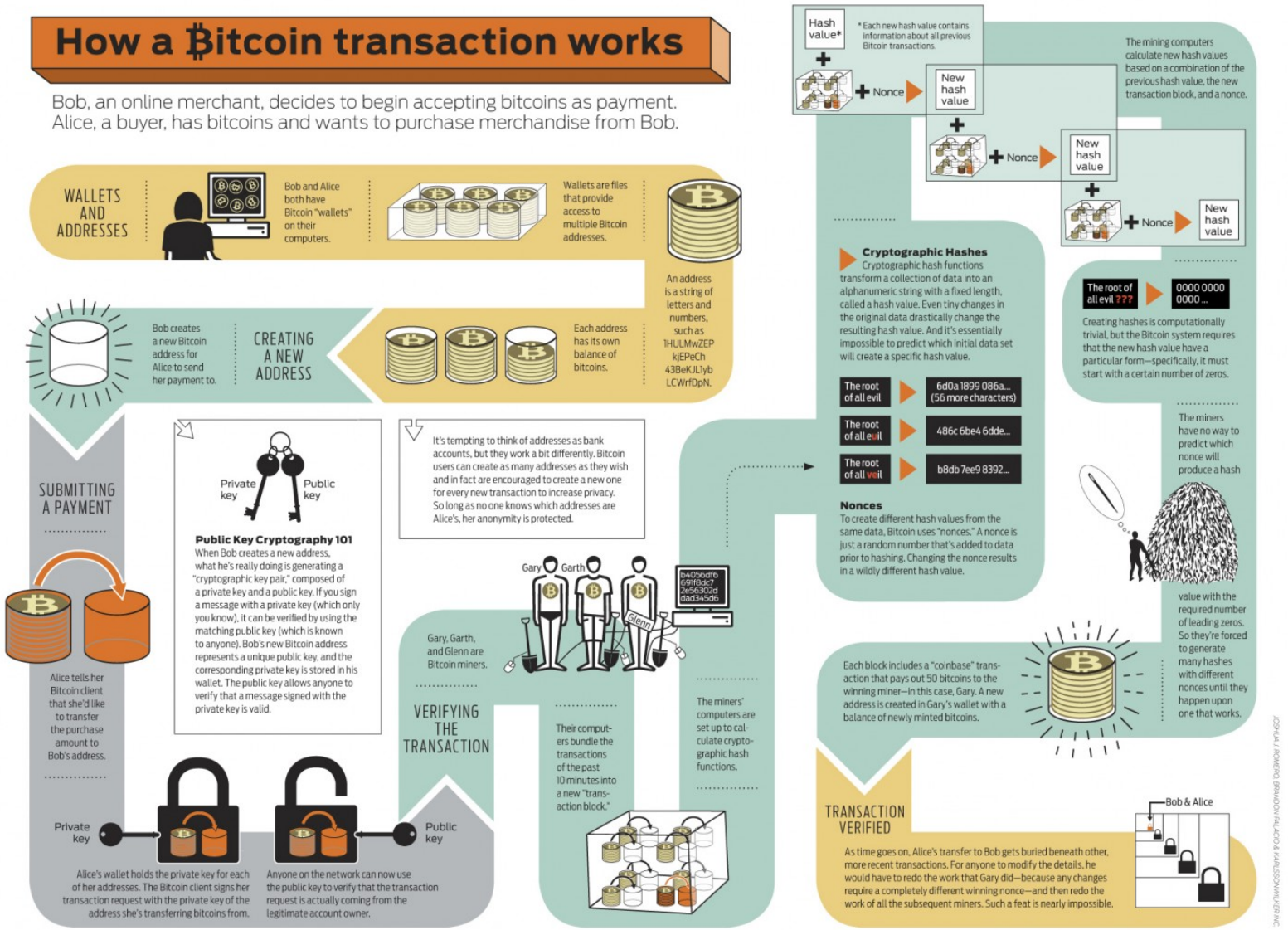
Outline of the transfer protocol

Here's how Alice transfers a Bitcoin to Bob:

1. Alice creates and signs a transaction request giving the coin to Bob.
2. The transaction is then broadcast to all of the miners.
3. Each miner first verifies the validity of the transaction by using its current most-recent copy of the database.
4. If valid, the miner attempts to create a new certified database incorporating the new transaction (along possibly with others) into the current database.
5. To certify a database requires solving a computationally-intensive puzzle.

Outline of the transfer protocol (cont.)

6. The puzzle consists of finding a nonce y such that the SHA-256 hash of the database together with y yields a hash value beginning with a long string of 0's.
7. A successful miner broadcasts the new database to all other miners.
8. Each miner upon receiving a new certified database discards all older ones and begins working with the newer one.
9. The system never reaches consensus, but the probability of a certified database being discarded in favor of another decreases exponentially over time.



From `http://lsvp.com/2013/03/28/how-bitcoin-works/`

Analysis

Why is consensus almost-certainly reached?

- Suppose two miners solve a puzzle simultaneously.
- Both broadcast their versions of the new database D and D'.
- Perhaps half of the miners work on D and half on D'.
- Most miners are likely attempting to incorporate Alice's transaction into a new database.
- Suppose some miner working on D solves the puzzle and sends out the new database D''.
- All miners receiving D'' discard the old D or D' and begin working on D''.
- Now an overwhelming majority of them believe D'' is the current database. They will only change their minds if a yet-longer certified database shows up.

Where's my money?

A good question to ask is, "Where is my money?".

It's obviously in the cloud, but where it is exactly is in the miners' computers.

Security relies on there being many honest miners.

Successful miners are currently rewarded with new Bitcoins, but as time goes on, the rewards are programmed to diminish.

What happens when miners no longer have the incentive to solve the computationally-intensive puzzles?

Other potential problems

There are other potential problems as well.

- What happens if Alice's private signing key gets compromised?
- What happens to Bitcoins that are lost?
- What happens if the puzzle turns out to be not as hard as expected?
- What happens if people turn their attention to a competing digital cash scheme?
- Is this another Ponzi scheme? Why or why not?
- Bitcoins have been compared to gold. Is that comparison valid?

Kerberos

Kerberos

Kerberos is a widely-used authentication system and protocol developed originally by M.I.T.'s Project Athena in the 1980's.

The protocol was named after the character Kerberos (or Cerberus) from Greek mythology which was a monstrous three-headed guard dog of Hades.

http://collectionsonline.lacma.org/mwebcgi/mweb.exe?request=record;id=12845;type=101

Simple authentication protocol

Alice and Bob want to communicate privately.

If they already share a private key K, they can just send encrypted messages to each other.

Problems with this approach:

1. Every time Alice uses K, she exposes it to possible cryptanalysis, so she really only wants to use it to establish a *session key* K_{ab} to encrypt her message to Bob.
2. Alice needs a different key for each different user she might wish to communicate with. In an N-party system, this could require $O(N^2)$ keys and becomes unwieldy.

Kerberos overview

Kerberos overcomes these problems by using a trusted server called the *Key Distribution Center (KDC)*.

Every user shares a key with the KDC.

When Alice wishes to talk to Bob, she asks the KDC to generate a session key K_{ab} for them to use.

The KDC uses Alice and Bob's private keys K_a and K_b for authentication and for the secure distribution of the session key K_{ab} to Alice and Bob.

Problems to overcome

The protocol must overcome several problems to be useful in practice:

- Network security is not assumed, so uses must never send their private keys over the network.
- Once Alice obtains K_{ab}, she needs a way of verifying that the other party holding K_{ab} is really Bob and not someone else pretending to be Bob.
- Users do not want to be constantly asked to provide their passwords, so a *single sign-on (SSO)* system is desirable.
- In a large system, the KDC could become a bottleneck, so it needs to be scalable.

Parties to the protocol

Four parties are involved in the basic protocol:

- The *authentication server (AS)*;
- The *ticket granting server (TGS)*;
- The *client, Alice in our examples*;
- The *service server (SS), Bob in our examples*.

The KDC contains the database of all keys and generally runs both the AS and the TGS.

Basic protocol

At a high level, the basic protocol consists of three phases:

1. Alice authenticates herself to the AS and receives a *ticket granting ticket (TGT)* in return.
2. Alice presents a TGT to the TGS to obtain an *Alice-to-Bob ticket*.
3. Alice presents the Alice-to-Bob ticket to Bob in order to obtain service.

Alice only uses her private key in step 1. The TGT obtained in step 1 contains a *client/TGS session key* that is used for securely communicating with the TGS in step 2.

Phase 1: Obtaining a TGT

Alice authenticates herself to AS and obtains a TGT.

- Alice sends a cleartext message with her ID "a" to the AS.
- The AS obtains Alice's secret key K_a from the database and sends back two messages:
  1. Message A: A Client/TGS session key $K_{a,\text{TGS}}$, encrypted with K_a.
  2. Message B: A TGT (Alice's ID, her IP address, expiration time, $K_{a,\text{TGS}}$), encrypted with K_{TGS}.
- Alice decrypts message A to obtain $K_{a,\text{TGS}}$. She is unable to decrypt message B.

Phase 2: Obtaining an A-to-B ticket

Alice uses her TGT to obtain an Alice-to-Bob ticket (A-to-B).

- Alice sends two messages to the TGS:
  1. Message C: (Message B, Bob's ID).
  2. Message D: (Alice's ID, timestamp), encrypted with $K_{a,\mathrm{TGS}}$.
- The TGS retrieves message B from message C and decrypts it to get $K_{a,\mathrm{TGS}}$, which it then uses to decrypt message D. It checks Alice's ID and IP address, generates a session key K_{ab} and then sends two messages to Alice:
  1. Message E: A-to-B ticket = (Alice's ID, her IP address, expiration time, $K_{a,b}$), encrypted using K_b.
  2. Message F: $K_{a,b}$, encrypted using $K_{a,\mathrm{TGS}}$.

Phase 3: Authenticating herself to Bob

Alice uses her A-to-B ticket to authorize herself to Bob.

- Alice sends two messages to Bob:
  1. Her A-to-B ticket, which she received from TGS as Message E.
  2. Message G: An authenticator = (Alice ID, timestamp), encrypted with $K_{a,b}$.
- Bob decrypts the ticket to retrieve $K_{a,b}$, which he uses it to decrypt the authenticator. He sends the following message to Alice:
  1. Message H: (1 + timestamp from the authenticator), encrypted with $K_{a,b}$.
- Alice decrypts and checks message H for correctness.

Use in practice

Tickets have a relatively long lifetime and can be used many times.

Authenticators have a relatively short lifetime and can be used only once.

The latest protocol has additional security enhancements beyond those described here.

Advantages of Kerberos

- Passwords aren't exposed to eavesdropping.
- Password is only typed to the local workstation.
  - It never travels over the network.
  - It is never transmitted to a remote server.
- Password guessing is more difficult.
- Single sign-on.
  - More convenient: only one password, entered once.
  - Users may be less likely to store passwords.
- Stolen tickets hard to reuse.
  - Need authenticator as well, which can't be reused.
- Much easier to effectively secure a small set of limited access machines (the KDC).
- Easier to recover from host compromises.
- Centralized user account administration.

Drawbacks and Limitations

- Kerberos server can impersonate anyone.
- KDC is a single point of failure.
  - Can have replicated KDC's.
- KDC could be a performance bottleneck.
  - Everyone needs to communicate with it frequently.
  - Not a practical concern these days.
  - Having multiple KDC's alleviates the problem.
- If local workstation is compromised, user's password could be stolen.
  - Only use a desktop machine or laptop that you trust.
  - Use hardware token pre-authentication.
- Kerberos vulnerable to password guessing attacks.
  - Choose good passwords!
  - Use hardware pre-authentication.
    - Hardware tokens, Smart cards etc.