CSE 565 Computer Security
Fall 2018

Lecture 23: Privacy Enhancing Technologies

Department of Computer Science and Engineering
University at Buffalo
Lecture Outline

- Electronic cash
  - anonymous spending
  - prevention of cheating
- Anonymous credentials and access control
- Secure voting
- Search over encrypted data
- Computation over encrypted data
As we perform many transactions in electronic form, there is a need for electronic money—check and credit cards leave trails—can we have an equivalent of anonymous cash?

Properties of cash

- it is anonymous and untraceable
- it can be used off-line, not connected to a bank
- it is transferable
- it has different denominations, and one can make change with it
- it can be used only once (or stolen)
Electronic Cash

- Can we design digital cash with similar properties that can be sent through computer networks?

- Let’s start with a very simple protocol – Protocol 0
  - the bank gives Alice a note for $10 and subtracts $10 from her bank account
  - Alice spends the note with a merchant
  - the merchant deposits the note in this bank account
  - the merchant’s bank clears the note with Alice’s bank

- This protocol has many problems
  - what exactly?
Electronic Cash

• We solve these problems using digital cash due to Chaum and others

• First let’s see how to make e-cash anonymous

• Suppose the bank has an RSA key (with \( pk = (n, e) \) and \( sk = d \))

• We’ll need to use RSA blind signatures:
  
  – given message \( m < n \) to be privately signed, choose random \( r < n \) and compute \( m' = m \cdot r^e \mod n \)
  
  – obtain plain RSA signature on \( m' \), where \( \text{sig}(m') = (m')^d \mod n \)
  
  – recover \( \text{sig}(m) \) from \( \text{sig}(m') \) by computing \( \text{sig}(m')r^{-1} \mod n \)
Electronic Cash

- **Protocol 1**
  - Alice makes 100 anonymous coins for $10 each
  - she blinds each coin and gives them all to the bank
  - the bank asks Alice to open randomly chosen 99 coins and verifies that each coin is for $10
  - the bank signs the last unopened coin, returns it to Alice, and deducts $10 from her account
  - Alice unblinds the signed coin and spends it with a merchant
  - the merchant verifies the bank’s signature to make sure it’s valid
  - the merchant takes the coin to his bank, which verifies the signature and adds $10 to the merchant’s account
• The technique for preventing cheating where a user sends a set of items and is asked to open a randomly chosen subset of them is called cut-and-choose

• Alice remains anonymous in Protocol 1 when the merchant deposits the coin as long as each value of $m$ cannot be linked to Alice

• Protocol 1 allows Alice to be anonymous, but a coin can still be spent more than once by Alice and the merchant
  – this is called double spending

• To eliminate the problem, we’ll require the bank to keep track of all spent coins
  – now we need to make sure that each coin is unique
Electronic Cash

- The bank now maintains a database of coin serial numbers it has seen
  - each coin \( m \) is formed as \( S||v \), where \( S \) is a randomly chosen serial number of the coin and \( v \) is its denomination
  - Alice must choose \( S \) to be long enough to make the chance of another person choosing it negligible

- Protocol 2
  - Alice prepares 100 $10 coins using a different serial number for each
  - she blinds all coins and gives them to the bank
  - the bank asks her to open 99 coin and checks whether they are properly formed
  - the bank signs the remaining coin and deducts $10 from Alice’s account
• **Protocol 2 (cont.)**
  
  – Alice unblinds the signed coin and spends it with a merchant
  
  – the merchant checks the signature to make sure the coin is valid
  
  – the merchant takes the coin to its bank, which first verifies the signature on the coin
  
  – the merchant’s bank also checks the database to make sure a coin with this serial number hasn’t been previously spent
  
  – if this hasn’t happened, the bank accepts the coin and adds $10 to the merchant’s account; and the bank rejects it otherwise

• This protocol protects the bank from cheating, but it doesn’t identify double spenders
• When cheating is detected, we want to be able to tell who is at fault
  – if Alice is trying to spend the same coin with more than one merchant, we want her to loose anonymity
  – if the merchant is trying to deposit Alice’s coin more than once, we want the bank to know that the merchant is cheating

• To be able to identify Alice when she double spends, we need to encode her identity into the coin
**Electronic Cash**

- **Protocol 3**
  - Alice prepares 100 $10 coins as follows and gives them blinded to the bank
    - on each coin she writes a random serial number $S$ and 100 pairs of identity strings $(I_{1L}, I_{1R}), \ldots, (I_{100L}, I_{100R})$
    - each part is a commitment that Alice can be asked to open
    - opened pair $(I_{iL}, I_{iR})$ reveals Alice’s identity, but two halves from different pairs $(I_{iL}, I_{jR})$ don’t
    - for example, such halves can be formed as
      \[
      I_{iL} = R_i, \quad I_{iR} = R_i \oplus \text{“Alice”}
      \]
      where $R_i$ is a randomly chosen string
  - the bank asks Alice to open 99 coins and verifies the contents
• Protocol 3 (cont.)

– the bank signs the last coin and deducts $10 from Alice’s account

– Alice unblinds the signed coin and spends it with a merchant, who verifies the bank’s signature

– the merchant asks Alice to randomly reveal either the left or right half of each of the 100 identity strings (of merchant’s choosing)

– Alice reveals them

– the merchant takes the coin to his bank, which verifies the signature and checks the database for the serial number

– if the serial number is not found, the bank credits the merchant $10 and records the coin in its database (including all opened identity string halves)
Protocol 3 (cont.)

- if the serial number is in the database, the bank rejects the coin
  - it compares the 100 identity strings on the coin with those in the database
  - if the opened sets are the same, the bank knows the merchant is double spending
  - if they differ, Alice spent her coin with a second merchant
  - in this case, the bank finds a pair \((I_{jL}, I_{jR})\) both halves of which are opened and identifies Alice

- Cheating by both users and merchants is now prevented
What properties does Protocol 3 have?

- Alice cannot cheat by double spending as she will be detected
- she can create a bad identity string with 1/100 success probability
- she cannot change the serial number because the bank’s signature will no longer be valid
- if the merchant cheats, he will be caught and Alice will not be implicated
- if Alice and the merchant conspire, they still cannot get the payment more than once
- can Mallory copy Alice’s coin and spend it first?
  * yes, and Alice might not even know about it
  * furthermore, if Mallory spends it twice, Alice goes to jail
• **Properties of Protocol 3** (cont.)

  – Mallory could eavesdrop on communication between Alice and the merchant and deposit the money (as a merchant) before the merchant does it
    • when the merchant tries to deposit it, he will be found as a cheater
  – thus, Alice and the merchant must protect their digital cash as if it were cash
    – it must be encrypted when sent across the Internet
    – finally, this e-cash is not transferable and one cannot make change with it
Let’s look at the **performance of Protocol 3**

- suppose that the bank cannot tolerate cheating with probability higher than 0.1% (1/1000)

- this means that Alice must initially produce 1000 coins, 999 of which the bank opens

- we also want the probability that Alice is not identified after spending a coin at two merchants to be low
  - suppose Alice includes 10 identity pairs in each coin
  - if two merchants choose their halves from each pair at random, the probability that they are exactly the same is $2^{-10}$
  - double spending Alice won’t be identified with probability $2^{-10}$

- so what is the communication and computation overhead?
Performance of Protocol 3 (cont.)

- let’s say that we use a hash function to produce commitments
- if each hash is 160 bits, 20 identity halves take 400 bytes
- 1000 coins amount to at least 400KB
- furthermore, the server’s work includes thousands operations per coin issued
- plus, all identity halves must be kept in the database of spent coins
- is there a way to do better?

The answer is yes, modern designs perform better
Electronic Cash

• Some other e-cash schemes are:
  – due to Brands (90s)
  – due to Camenisch, Lysyanskaya, and others (00s)

• Their features:
  – they rely on the difficulty of computing discrete log in groups modulo a prime
  – they avoid expensive cut-and-choose techniques
    • instead, a user can convince the bank that the coin is well formed through other means
    • often this is done using zero-knowledge proofs of knowledge
Zero-Knowledge Proofs of Knowledge

- Recall that in zero-knowledge proofs of knowledge (ZKPK)
  - knowledge of a secret is required to successfully produce a proof
  - no information about the secret is revealed during the protocol
- ZKPKs exist for many types of problems including all NP-languages
  - many of such solutions, however, are not efficient and mainly of theoretical interest
  - but efficient ZKPKs exist for several statements based on discrete logarithms
Anonymous Credentials

• Both efficient e-cash and anonymous access control can be implemented using so-called signatures with protocols.

• In such a signature scheme, a user can
  – obtain a signature $\text{sig}(m)$ on message $m$ by revealing only a commitment to it $\text{com}(m)$
    • a commitment scheme is expected to have hiding and binding properties
  – randomize signature $\text{sig}(m)$
    • different showings of the signature cannot be linked together
  – prove statements about the signed value in zero-knowledge

• Often the signature scheme allows several messages to be included in a single signature.
Anonymous Access Control

- To permit **anonymous authentication**, a user obtains authority’s certification in the form of a signature on some attributes
  - the authority can know all attributes or have only partial information about them
  - the attributes that should remain hidden from the authority are sent in the form of a commitment
- Each time such credentials are used, the user
  - needs to randomize them
  - prove that the signed values satisfy the access control policy
Anonymous Access Control

• Examples
  – the user can prove that she is over 21 without revealing the birth date (or anything else)
  – the user can prove that she is a student member and the expiration date is some time in the future

• One significant issue with using anonymous credentials in a commercial setting is prevention of duplicating user credentials
Anonymous Access Control

• Solutions to the problem of credential duplication include:
  – incorporating sensitive information into each credential the knowledge of which must be shown upon each use
  – restricting the number of simultaneous uses of a credential
  – issuing one-time credentials that can be exchanged for a new token upon each use

• Certain other techniques allow a user to be anonymous with the ability to uncover the user’s identity under exceptional circumstances
• Efficient e-cash can be built using anonymous credentials as follows:
  – a user forms commitment $\text{com}(s)$, where $s$ is random serial number
  – the bank produces a coin as $\text{sig}(s, v, id)$ using user’s $id$ and coin’s denomination $v$
  – when user spends the coin, she reveals $s$, $v$, and a function of her $id$ to the merchant
  – the function is such that
    • if evaluated on a single point, it reveals no information about $id$
    • but when evaluated on more than one points, the $id$ can be easily computed
  – double spending by the user reveals the user’s identity, but depositing twice with the same numbers makes the merchant guilty
• **Bitcoin** solved the chicken and an egg problem in adopting digital cash by adopting a different model
  
  — real banks are not a part of the protocol and digital transactions
  
  — **blockchain** is a mechanism for distributed consensus
    
    • previous transactions are recorded and stored at many participants
    
    • miners have to perform work to form a new block to be appended to the blockchain
    
    • the concept of the proof of work holds it all together
    
    • a blockchain can branch, but only one branch eventually survives
  
• Users are pseudonymous because signing keys (used for transactions) are not linked to real identities
Preserving privacy of voters is crucial for fair outcome of elections

Secure elections should have at least the following properties:

1. only registered voters can vote
2. no person can vote more than once
3. no one can determine for whom anyone else voted
4. every voter can make sure that his vote has been counted
5. no person can duplicate any other person’s vote
6. no person can change any other person’s vote undetected

How can we achieve this?
Secure Elections

- Suppose there is Central Tabulating Facility CTF

- **Protocol 1**
  - each voter encrypts vote with CTF’s public key and mails it to CTF
  - CTF decrypts the votes, tabulates them, and publishes the results

- What desired properties are achieved?

- **Protocol 2**: use signatures
  - each voter signs her vote with her private signing key
  - each voter encrypts her signed vote with CTF’s public key and mails it to CTF
  - CTF decrypts the votes, checks signatures, tabulates them, and publishes the results
Secure Elections

- **Protocol 3**: use blinding
  - each voter prepares 100 sets of messages
    - each set contains a valid vote for each possible outcome
    - each message contains a random 20-digit number
  - each voter blinds each message individually and sends them to CTF
  - CTF checks to make sure the voter hasn’t voted, asks the voter to open 99 sets of messages, checks them, and signs messages in the remaining set
  - voter unblinds signatures, chooses a vote, encrypts it with CTF’s (encryption) key and mails it to CTF
  - CTF decrypts the votes, checks signatures, checks for duplicates, tabulates votes, and publishes the results
The next protocol finally achieves all desired properties and the two additional properties:

7. if a voter finds that his vote is miscounted, he can correct the problem without compromising secrecy of his ballot

8. a voter can redicide and cast a new vote within allocated time frame

**Protocol 4:** use anonymous IDs and one-time encryption keys

- CTF publishes a list of eligible voters who intend to vote
- each voter receives a unique number $I$ not known to CTF
- each voter generates public-key encryption pair $(pk, sk)$ and sends $Enc_{pk}(I; v)$, where $v$ is the vote, to CTF
- CTF publishes receipt $Enc_{pk}(I; v)$
Secure Elections

- **Protocol 4 (cont.):**
  - each voter sends $I, sk$ to CTF
  - CTF decrypts votes and at the end of election publishes results: for each vote $v$ the list of all $\text{Enc}_{pk}(I; v)$ that contained it
  - if a voter sees error, she objects by sending $I, \text{Enc}_{pk}(I; v), sk$ to CTF
  - if voter wants to change vote from $v$ to $v'$, he sends $I, \text{Enc}_{pk}(I; v'), sk$ to CTF

- Malicious CTF still has some powers:
  - forging votes for people who intended to vote, but didn’t
  - neglecting to count Alice’s vote and claiming that Alice never voted
  - are better solutions possible?
Alice has a set of documents that she stores on untrusted server Bob
  – e.g., mobile user storing email messages
  – for privacy reasons, the documents are encrypted

Alice would like to retrieve only documents that contain the word $W$

Each document is divided into fixed length blocks called words

Search can be performed with or without an index
  – for each word $W$ of interest, the index will list all documents containing it
  – using index is faster with large documents, but adds overhead for its maintenance
• Approach 1

– the solution uses a pseudo-random function $F$: on input key $k$ and message $m$, $F(k, m)$ outputs a pseudo-random sequence

– Alice wants to encrypt a document consisting of $n$-bit words $W_1, \ldots, W_\ell$

– Alice generates a sequence of $(n - m)$-bit pseudorandom values $S_1, \ldots, S_\ell$

– to encrypt $W_i$, Alice sets $T_i = \langle S_i, F(k_i, S_i) \rangle$ and outputs ciphertext $C_i = W_i \oplus T_i$
Search on Encrypted Data without Index

- **Approach 1 (cont.)**
  - to decrypt, Alice reproduces $S_1, \ldots, S_\ell$, computes all $F(k_i, S_i)$, and recovers $W_1, \ldots, W_\ell$
  - when Alice wants to search for $W$, she tells Bob $W$ and $k_i$ for each location where $W$ may occur
  - Bob searches for $W$ by checking whether $C_i \oplus W$ is of the form $\langle s, F(k_i, s) \rangle$ for some $s$

- What problems does this solution have?
  - Alice must either know at what positions $W$ is located
  - or Alice reveals all keys $k_i$ (potentially revealing entire document)
• **Approach 2**: controlled searching

  – Alice chooses a random secret key $k$

  – during encryption of $W_i$, Alice uses $k_i = F'(k, W_i)$, where $F'$ is also pseudo-random function

  – now when she wants Bob to search for $W$, she reveals $F'(k, W)$ and $W$ to him

  – Bob can identify all locations where $W$ occurs, but learns nothing about locations $i$ where $W_i \neq W$
• **Approach 3**: support for hidden searches

  – Alice does not want to reveal $W$ to Bob during searches

  – Alice now chooses another secret key $k'$

  – Alice encrypts each word $W_i$ with **deterministic** encryption as
    \[ X_i = E(k', W_i) \]

  – the rest is performed as before:
    
    • Alice forms $C_i = X_i \oplus T_i$, where $T_i = \langle S_i, F(k_i, S_i) \rangle$ and $k_i = F'(k, X_i)$
    
    • to search for $W$, Alice computes $X = E(k', W)$ and $k_X = F'(k, X)$ and sends $X, k_X$ to Bob
• **Approach 4**: final solution
  
  – there is one problem with previous solution
    • it is no longer possible to decrypt the document given only its ciphertext
    • to compute $F(k_i, S_i)$, one has to know $W_i$ it encrypts since $k_i$ is a function of $X_i = E(k', W_i)$
  
  – the solution is to compute $k_i$ from the “recoverable” portion of $X_i$
    • during encryption, split $X_i = E(k', W_i)$ into two parts $X_i = \langle L_i, R_i \rangle$, where $L_i$ is $n - m$ bits long
    • compute $k_i$ as $F'(k, L_i)$
    • now $L_i$ is computed during decryption using XOR and then $k_i$ is computed using it
Search on Encrypted Data without Index

- Approach 4

\[
\begin{align*}
W_i & \xrightarrow{E} E(k', W_i) \\
E(k', W_i) & \xrightarrow{C_i} C_i
\end{align*}
\]

- A number of extensions are possible such as more advanced queries
Technical solutions to privacy are numerous

- in certain applications with want to combine anonymity with accountability
- in other applications we seek to protect private information

Work on privacy and anonymity started in early 80s and continues to date

- efficient constructions for applications such as e-cash, anonymous credentials, etc. are known
- there is always room for improvement