Show, But Don’t Tell

John J. Mikucki

October 18, 2001

Slides to be available at http://www.cs.rit.edu/~j
Goals

1. Understand concept of interactive and zero-knowledge proofs (ZKP)

2. Review examples of ZKP’s

3. Understand bit-commitment

4. Discuss ZKP uses
Why Zero Knowledge?

1. Stock market predictions

2. Bit Commitment and trash collection

3. Identity over insecure channel

4. Any time you want to show knowledge without sharing it.
Basics

Prover

– Entity in our proofs who will do the ‘proving’
– Usually computationally unbounded

Verifier

– Entity who will be issuing challenges to prover

Key behavior: At the end of each proof round, denotes response indicating ‘acceptance’ of the round.
The Subwayman

1. Starting from the entrance, Paul advances beyond it, out of sight.

2. Paul flips a coin to choose a fork at random. He turns and heads in the opposite direction of it, out of sight.

3. We enter the tunnel and wait at the fork.

4. We flip a coin to randomly choose a tunnel. We tell Paul to randomly flip a coin to randomly choose a tunnel. We tell Paul to emerge from that tunnel.

5. Paul returns.
6. We note whether Paul returns from the requested activity.

7. Return to the entrance and repeat.
More Definitions

View

–also called a *transcript*

–Totality of communication (ie a complete log) between the prover and the verifier during the proving process.

–*Key behavior*: A verifier ‘sees’ a view. Verifiers are also called *observers*. 
Completeness

Given an honest prover and verifier, the protocol succeeds with overwhelming probability.

Soundness

There is a polynomial-time algorithm $M$ such that if an adversary prover $P'$ can, with non-negligible probability, successfully complete the protocol with $V$, then $M$ allows $P'$ to figure out a piece of information functionally equivalent to the genuine prover $P$’s secret.
Proof of Knowledge

An interactive proof is said to be a proof of knowledge if it has both the properties of completeness and soundness.

Simulator

A method or procedure that generates fake (generated with the prover) views indistinguishable from a genuine (interacted with the prover) view of a proof.

Zero-knowledge Property

A proof of knowledge has the zero-knowledge property if there exists a simulator for the proof.
Zero Knowledge Proof

– should always convince an honest verifier when the prover knows the secret;

– should almost never convince an honest verifier when the prover doesn’t know the secret;

– should not convey any useful knowledge to any observer except the fact that the prover knows (or doesn’t) the secret.
Perfect Zero Knowledge

For a decision problem \( \Pi \), a prover \( P \), and a verifier \( V \), an algorithm \( V^* \), there exist two sets, \( \tau(V^*) \) and \( F(V^*) \) of genuine and forged transcripts, respectively.

A proof is *perfect zero knowledge* if

\[
\forall T \in \tau(V^*), P(\tau(V^*)) = P(F(V^*))
\]
Other Zero Knowledge

*Computational Zero Knowledge*

An observer restricted to probabilistic polynomial-time can not distinguish real from simulated transcripts.

*Statistical Zero Knowledge*

There is a negligible difference between the probability distribution of genuine views $P_r$ and the probability distribution of simulated views, $P_F$. 
Welsh Map

The map on the wall has 5 major cities on it. The Welsh names are horrendous to spell, so Phil denotes them by the numbers 1 to 5. Phil and Vern’s maps are labeled with city names 1 to 5. They denote the wall map $G_1$ and their map $G_2$. Phil knows the secret $\sigma = \{5, 4, 3, 2, 1\}$ maps the nodes 5, 4, 3, 2, 1 on the wall map to the nodes 1, 2, 3, 4, 5 on the graded maps. He will prove the isomorphism to Vernor with it away.
Graph Isomorphism Protocol

1. Phil selects a random permutation $\pi$ of $\{1, \ldots, 5\}$ and
   constructs a new map, $H$, that is the image of $G_1$ under $\pi$, and
   sends this map to Vernor.

2. Vernor chooses a random integer $i \in \{1, 2\}$ and
   sends this integer to Phil.

3. Phil computes a permutation $\rho$ of $\{1, \ldots, 5\}$ such
   that $\rho$ is the image of $G_i$ under $P$. If $i = 1$, Phil uses
   $\rho = \pi$. If $i = 2$, Phil uses $\rho = \sigma \circ \pi$, where $\sigma$
   is a fixed permutation. Then $G_1 = \rho(G_2)$. 
4. Vernor checks that $H$ is the image of $G_i$. 
View

Our transcript is: \( T = \{(G_1, G_2); (H_1, i_1, \rho_1), \ldots \} \) So we have:

\[
T = \{ (\{1, 2, 3, 4, 5\}, \{5, 4, 3, 2, 1\}); \\
(\{1, 4, 3, 2, 5\}, 1, \{1, 4, 3, 2, 5\}); \\
(\{2, 4, 1, 3, 5\}, 2, \{4, 2, 5, 3, 1\}); \\
\ldots \\
(\{H_n, i_n, \rho_n\}) \}
\]
Forgery Algorithm

1. \( T_0 = (G_1, G_2) \)

2. Randomly select \( i_b \in \{1, 2\} \).

3. Create a random permutation \( H_b = \rho_b \circ G_b \).

4. Add \( (H_b, i_b, \rho_b) \) to \( T \).

5. Repeat until the desired transcript length has been reached (usually \( n \) rounds, where \( n = |G^*| \)).
Forging Type I Rounds

1. \( T = \{([1, 2, 3, 4, 5], [5, 4, 3, 2, 1])\} \)

2. We select \( i = 1 \).

3. Create random permutation \( \rho_b = \{1, 3, 5, 4, 2\} \).
   generate \( H_b = \rho_b \circ G_i = \{5, 3, 1, 4, 2\} \).

4. Add to \( T \), which becomes \( \{([1, 2, 3, 4, 5], [5, 4, 3, 2, 1]), ([1, 3, 5, 4, 2], 1, [5, 3, 1, 4, 2])\} \).
Forging Type II Rounds

1. Select \( i = 2 \).

2. Generate another random permutation \( \rho_b = \{2, 3, 4, 1, 5\} \). This gives us \( H_b = \rho_b \circ G_i = \{2, 3, 4, 1, 5\} \).

3. Append \( (\{2, 3, 4, 1, 5\}, 2, \{2, 3, 4, 1, 5\}) \) to \( T \).
Quadratic Residues Protocol

Pete will prove, Vinnie will verify.

1. Pete chooses a random $v \in \mathbb{Z}_n^*$, and computes $y = v^2 \mod n$. He sends $y$ to Vinnie.

2. Vinnie chooses $i \in 0, 1$ and sends it to Pete.

3. Pete computes $z = u^i v \mod n$, ($u$ being $\sqrt{v} \mod n$) and sends $z$ to Vinnie.

4. Vinnie verifies that $z^2 \equiv x^i y \mod n$. 
5. Repeat from step 1 $\log_2 n$ times or as needed.
\[ n = p \cdot q = 3 \cdot 11 = 33. \]

\[
\mathbb{Z}_{33}^* = \begin{array}{cccc}
  \begin{array}{cccc}
    i & i^2 \mod 33 & i & i^2 \mod 33 \\
    1 & 1 & 17 & 16 \\
    2 & 4 & 19 & 31 \\
    4 & 16 & 20 & 4 \\
    5 & 25 & 23 & 1 \\
    7 & 16 & 25 & 31 \\
    8 & 31 & 26 & 16 \\
    10 & 1 & 28 & 25 \\
    13 & 4 & 29 & 16 \\
    14 & 16 & 31 & 4 \\
    16 & 25 & 32 & 1 \\
  \end{array}
\end{array}
\]
We select \( x \in QR(33) = 31 \).

**Round 1**

1. Pete picks \( y = v^2 \mod 33 = 4 \), and sends it to Vinnie.

2. Vic selects \( i = 1 \) and sends it to Pete.

3. Pete computes \( u = \sqrt{v} \mod n = 8 \), and uses it to compute
   \[ z = u^i v \mod n = 8^1 \cdot 4 \mod 33 = 32 \]
   Pete sends \( z \) to Vinnie.

4. Vinnie verifies that \( z^2 = 1 \equiv 8^1 \cdot 4 \mod 33 = 1 \).
Round 2

1. Pete picks a new $v = 16$, computes $y = v^2 = 25$, and sends it to Vinnie.

2. Vic chooses $i = 0$, and sends it to Pete.

3. Pete computes $u = \sqrt{v} \mod n = 4$, and then $z = u^i \mod 33 = 16$. He sends $z$ to Vinnie.

4. Vinnie checks that $z^2 = 25 \equiv 31 \mod 33 = 2$.
Turing machines

– *deterministic* TM’s can solve problems in $P$ and verify problems in $NP$.

– *non-deterministic* TM’s can solve problems in $NP$
Bit Commitment

–We want to commit to a string of bits without having to mail them.

–Safes are inconvenient to mail; we need a better way.

–Two examples: symmetric-key and hash-based bit commitment schemes. A’s will commit to bits for B’s.
Symmetric-Key Bit Commitment

1. Ben generates a random bitstring $R$ and sends it to Ann.

2. Ann takes Ben’s string $R$, and $B$ (the bits she’s committed to), puts them into a message, and encrypts the resulting vector with a random key $K$. Ann sends the resulting packet (both $R$ and $B$) to Bob. The bits are now ‘committed’.

3. To ’open the safe’, Ann sends Ben $K$.

4. Ben decrypts the packet he got from Ann. He verifies that the bit-string $R$ is in the packet, along with $B$. 
Discussion

- Works because Ann and Ben are computationally bounded in $P$-space. Using encryption based on exponential-time problems, Ben and Ann can’t effectively cheat.

- There’s no way to ensure that two infinitely powerful agents don’t cheat with a purely mental protocol.
Hash-based bit commitment

1. Ann generates two random strings, $R_1$ and $R_2$.

2. Ann concatenates $R_1$, $R_2$, and the bitstring she’s encrypting, calls it $M$, and hashes it.

3. Ann sends the hash and one of the random strings to Ben.

4. When the time is right to open the message, Ann sends $M$.

5. Ben hashes $M$ and checks that it’s hash was the same as the one Ann had previously sent him.
Analysis

The protocol is secure if the hash function is good. We expect to succeed at finding another string that hashes identically. Bob’s sole responsibility is correctly hashing the data he receives; he’s almost completely passive.
Non-interactive proofs

Feed a hash function a constant, publicly available bitstring related to the starting conditions of the proof (sets of parameters $n$, $p$, $q$, etc). The hash is a bitstring with nice properties:

1. Bits are pseudo-random in that we didn’t know what they were before we generated them, and we can’t generate starting conditions from a hash.

2. Bits are deterministic in that we can always generate them by feeding the (specified) hash function the same input.
Because the prover can’t know the generated hash generation, they’re functionally equivalent to verifying random bits for the prover.
Application

Feige-Fiat-Shamir Identity Proof

Guillou-Quisquater Proof of Identity

Schnorr’s Authentication Protocol
Feige-Fiat-Shamir Setup

Paula will prove, Verona will verify, and trusty Al will

Al chooses random modulus \( n = pq \) where \( p, q \in \mathcal{P} \). He chooses \( v \in QR(n) \) and for which \( \exists v^{-1} \mod n \). \( v \) is Paula's key. \( s \), the smallest number such that \( s = \sqrt{\frac{x}{v}} \mod n \), is Paula's private key.
**FFS protocol**

1. Paula picks a random number $r, 1 \leq r < n$. As her first move, she computes $x = r^2 \mod n$ and sends it to Verona.

2. Verona sends a random $b \in \{0, 1\}$ to Paula.

3. Paula sends back

   $$
   \begin{cases} 
   b = 0 & r \\
   b = 1 & y = rs \mod n 
   \end{cases}
   $$
4. Verona verifies that

\[
\begin{cases} 
  b = 0 & x = r^2 \mod n \\
  b = 1 & x = y^2v \mod n
\end{cases}
\]

5. Repeat from step 2 until satisfied of authenticity.

N should be no less than 512 bits long to be secure. Further, there must be a secure means of communicating s between Mike and Paula.
GQ Setup

Pat will prove, Vaughn will verify.

Vaughn makes public a fixed exponent $v$ and modulus $n$, a public credential $J$. Pat computes her private key $B$ such that $JB^v = 1 \mod n$. Once that’s done, they can use the process whenever Pat needs to authenticate to Vaughn.
**GQ Protocol**

1. Pat picks a random $r, 1 < r < n$, computes $T = J^r B^r$ and sends Vaughn $(J, r, T)$.

2. Vaughn picks a random $d, 0 < d < v$, and sends it to Pat.

3. Pat computes $D = r B^d \mod n$ and sends it to Vaughn.

4. Vaughn computes $T' = D^v J^d \mod n$. If $T = T'$, the protocol succeeds.
GQ Analysis

\[ T^l = D^v J^d = (r B^d)^v J^d = r^v B^{dv} J^d = r^v (J B^v)^d = r^v = 0 \]

The trick lies in the careful construction of \( B \).
Schnorr Setup

Pennie will prove, Vi will verify. Pennie takes the cuales of primes $p$ and $q$, selecting $q \in \mathcal{P}$ and $q | p - 1$. She then chooses an element $a \in \mathbb{Z}_p$ such that $a \neq 1$, $a^q = 1 \mod p$.

Pennie new generates a random private key $s$, $0 < s < q$, which she keeps secret, and calculates her public key $v = a^s \mod p$, which she publishes. Finally, we need a number $t$ of at least 72 bits.
Schnorr’s Protocol

1. Pennie randomly picks \( r, r < q \). She then computes \( a^r \mod p \). She sends \( x \) to Vi.

2. Vi sends Pennie a random number \( e, 0 < e < 2^t - 1 \).

3. Pennie sends \( y = (r + se) \mod q \) to Vi.

4. Vi checks that \( x = a^y v^e \mod p \).
Discussion

The real question is where the $t$ fits in. The answer provides the security. The difficulty of breaking Schnorr's $2^t$, I believe because in the final step both exponents are $e$ (see step 3).