6.857 Computer and Network Security October 22, 2002

Lecture Notes 13 : Palladium, Zero Knowledge

Lecturer: Ron Rivest *Scribe: Baratz/Gavacs/Sen/Sudan*

[These are the initial scribe notes. The final version will appear with updated figures. Namely, the figures will have larger fonts.]

1 Outline

- Palladium discussion
- Zero Knowledge Proofs

2 Palladium discussion

Prof. Rivest: What did people like / dislike about Palladium?
Student: I think it's interesting to think about the various other organizations that are affecting Palladium, like Hollywood, etc.
Student: I don't think Palladium is going to fly. They haven't really come up with a killer-app and the cost is going to be too high. What is the killer app? Movie and music distribution?

Prof. Rivest: Could movie distribution be the killer app? That really seems to be their driving motivation.
Student: It seems as though the only way they can justify this initiative is if they envision PCs becoming the center of a home theater system. Using PCs to control DVD players, TVs, etc.

Prof. Rivest: A very useful way of thinking about it is as a virtual embedded set top box.
Student: How can they use this system for DRM if it isn't physically tamper-resistant? Maybe due to the DMCA it would be illegal to install dual-ported memory. Hardware attacks could probably be carried out for hundreds of dollars or less. A movie could be extracted and then distributed.

Prof. Rivest: Besides DRM, what could this be used for?
Student: Possibly subscription services, software licensing, or piracy control.
Student: The whole TCPA framework provides a lot of functionality to enterprises.
Student: It seems as though the right-hand side of Palladium won't really be used that much and isn't robust enough to run complete applications like Word, etc.

Prof. Rivest: This reminds me of how we drew the distinction between user and kernel space, and then with Microsoft operating systems and plug-and-play people have been able to insert drivers, etc. into kernel space. Now all they've done is draw another line and are daring outsiders to cross that line. After a while all sorts of code will have found its way into the Palladium zone and then what do we do? Draw another line and make Palladium 2?

Prof. Rivest: Any other questions about Palladium or its applications. How many of you think Palladium isn't going to fly? A couple dozen people raise their hands. Why isn't it going to fly?
Student: Not enough perceived benefit to the user. I don't think the end user desire exists. Lack of motivation. Maybe government agencies and enterprises will be interested.

Prof. Rivest: It will be interesting to see how this rolls out: So one of the things Palladium does is burn in keys.

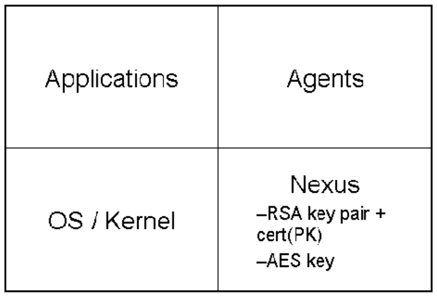

Figure 1: Palladium model. The keys shown here in the nexus are actually in a separate chip called the SSC.

PK represents the machine's identity. We don't necessarily want to sign everything with PK, since this will reveal machine-specific information that could compromise our privacy. One thing we could do is generate a new PK1 and send PK, cert(PK), and PK1 to a certificate authority (CA). The CA would then return cert(PK1). Basically the certificate says that the key belongs to a Palladium machine, but doesn't say which one. We have created an alias and provided anonymity.

The problem with this architecture is that the CA knows all the mappings from PK to PK1. This may be what you want, but if not, what you might really be after is a way to convince the CA that you have a Palladium machine with a PK/SK and a certificate, without actually revealing those items.

How can this be done? Enter Zero-Knowledge (ZK) Proofs:

3 Zero-Knowledge Proofs

In a zero-knowledge (ZK) proof, you are basically trying to convince someone that you know something without telling them what it is, such as the message m corresponding to a known public ciphertext c.

3.1 The General Set-up

P = Prover

wants to prove he knows m

V = Verifier

wants to learn if P knows m

$\xrightarrow{w=\text{witness}}$

$\xleftarrow{x=\text{random challenge}}$

$\xrightarrow{y=\text{response}}$ test (c, w, x, y).

Accept if OK, reject otherwise.

We want the protocol to satisfy the following properties:

1. If P does know m, then V always accepts. (Completeness)
2. If P does not know m, then P is unlikely to convince V. (Soundness)

Soundness may be shown by demonstrating that if P is prepared to respond to several different challenges (for the same witness), then in fact P must know (or must be able to easily compute) m.

We would also like to have that the protocol be *zero-knowledge*: the Verifier learns nothing (zero), aside from the fact that P knows m.

3.2 3-Colorability Example

We have an undirected graph with 5 vertices. Can we color the vertices with three colors so that no two adjacent vertices have the same color?

A given graph may have many possible colorings, only one coloring, or no colorings at all. Suppose I have a graph with a million vertices and I tell you I know how to color it using 3 colors. I want to convince you that the graph is 3-colorable without telling you what the coloring is. Is there some way I can convince a remote computer, the skeptic, that I know how to do the coloring without saying what the coloring is?

Consider the following exchange between the Prover (you, the person who knows the 3-coloring) and the Verifier (the remote computer you are trying to convince):

For a given graph coloring, there are 6 different permutations of the coloring possible: RGB, RBG, BRG, BGR, GBR, GRB. That is, you take one coloring and just permute the color assignments (e.g. from RGB to RBG, all vertices that are R are still R, all vertices that are G are now B, etc.). Again, this is just for one coloring that the Prover picks. We can represent each permutation as a sheet of paper with the graph and coloring on it (see Figure 4 below).

[Figure 4: Representation of the 6 permutations of a given graph 3-coloring]

Do t times: Prover: picks a sheet (graph with a coloring permutation) from a random pile (Verifier doesn't know which pile Prover picks) and puts it on the table covering up vertices with little stickies

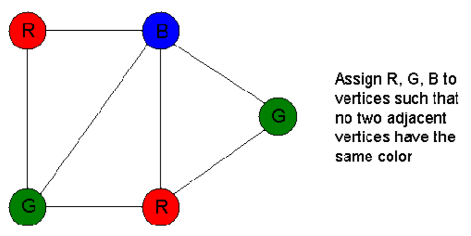
R
B
G
G
R
Assign R, G, B to vertices such that no two adjacent vertices have the same color

Figure 2: 3-coloring of an undirected graph with 5 vertices

Verifier: points to 2 adjacent vertices Prover: removes stickies from the two selected vertices Verifier: checks that two vertices have different colors (if not $\Rightarrow$ *not OK*)

Output: *OK*

This algorithm is polynomial in the size of the graph (V + E, where V is the set of vertices and E is the set of edges), assuming that t is bounded by a polynomial in $|V|$ and $|E|$.

What are the properties of this protocol?

- Completeness: if Prover knows coloring, Verifier always accepts
- Soundness: if Prover doesn't know coloring, he will be caught with significant probability.
  o Prob(Verifier picks "bad edge") $\geq \frac{1}{|E|}$
  o Prob(Verifier picks "good edge") $\leq 1 - \frac{1}{|E|}$
  o So in t trials: Prob (bad prover escapes detection) $\leq (1 - \frac{1}{|E|})^t \leq e^{\frac{-t}{|E|}}$

Since $e^x \geq 1 + x$, this probability is bounded above by $e^{-t/|E|}$, which for $t = |E|20$ is less than e^{-20}.

Q: If t is $|E| \times 20$, aren't you exposing more than the number of edges in the graph?
A: Yes, but each time you expose an edge it could be from any of the 6 piles (the Verifier doesn't know which pile the user picks)

We will try to show that the Verifier doesn't learn anything if this scheme is followed. The idea is that there is no correlation between pairs of colors revealed by each edge.

Q: Why can't the Prover cheat by just showing different colors for each pair of vertices requested by the Verifier (that is, choosing the vertex colors right before revealing them and that way guaranteeing

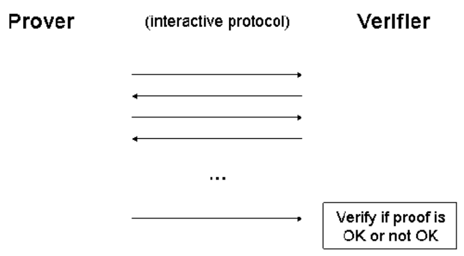

Figure 3: High-level overview of exchange between Prover and Verifier

that the coloring works)?
A: The Prover actually commits to a coloring before hand, so all he can do is remove the stickies and expose the vertex colors.

Q: Does the Prover need to know all possible colorings in this scheme?
A: No (look above). The Prover picks one coloring and just permutes the color assignments (so the coloring scheme actually remains the same).

Our informal proof of "zero-knowledge":

The Verifier gets a transcript of his conversation with the Prover and nothing more (transcript embodies all information obtained by the Verifier). We are assuming that the Prover takes the same amount of time to respond to each challenge (so, for instance, the Verifier can't learn anything extra based on the time taken for the Prover to respond).

The information we get from this protocol is:

This distribution of transcripts can be simulated by verifier, without Prover's help.

3.3 Proof using discrete logs

We now give another illustration of a zero-knowledge protocol. The goal of this protocol is for the Prover to convince the Verifier that he knows the discrete logarithm x of a public value (his public key) y.

Global public parameters: prime p, prime q dividing $p-1$, g of order q.

Public key of prover: $y = g^x \bmod p$

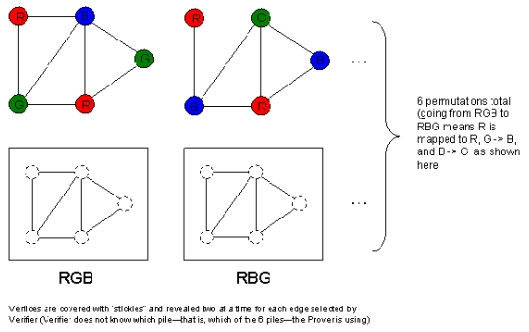
R
B
G
G
R
R
G
B
B
R
...
...
6 permutations total (going from RGB to RBG means R is mapped to R, G -> B, and B -> G as shown here
RGB
RBG
Vertices are covered with "stickies" and revealed two at a time for each edge selected by Verifier (Verifier does not know which pile—that is, which of the 6 piles—the Prover is using)

Figure 4: Representation of the 6 permutations of a given graph 3-coloring

Private key of prover: x s.t. $0 \leq x < q$

Do t times:

- Prover: picks $k \in Z_q$, sends $u = g^k \bmod p$
- Verifier: computes $b \in_R \{0, 1\}$, sends b to Prover
- Prover: sends $l = k + bx \bmod q$
- Verifier: computes $u \times y^b = g^l \bmod p$

Note that the Prover can always succeed in each round with probability 1/2, by guessing b first, then setting $u = g^l / y^b$.

This protocol is sound and complete. The Verifier will always accept an honest Prover's proof, and a cheating prover will be caught with significant probability in each round.

Now the claim is that the protocol is ZK. We can play this game all day long and there is no way the Verifier will learn any information about x. The Verifier just gets a huge list of u, b, and l during each iteration, but the claim again is that this tells us nothing about x.

To demonstrate our claim, we can generate transcripts (on our own) with the proper distribution:

- pick $b \in_R \{0, 1\}$
- pick $l \in_R Z_q$
- pick $u = g^l / y^b \bmod p$

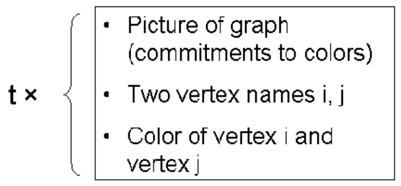
t ×
• Picture of graph (commitments to colors)
• Two vertex names i, j
• Color of vertex i and vertex j

Figure 5: Information gained by Verifier after t iterations of above communication protocol

This gives us a triple (u, b, l) which we claim is indistinguishable from the actual transcript generated by the Prover-Verifier protocol above. (This is for an honest verifier. If the verifier actually generates b in some other way, such as making it depend on u, then this simulation needs to be modified by filtering the output appropriately. Details omitted in this class. But in any case, the Verifier can simulate the distribution on transcripts, which proves that the protocol is ZK.)

A much more general and wonderful result is also known. Indeed, for any polynomial time program P, I can convince you (in zero knowledge) that I know x, y, z, etc. such that $P(x, y, z, \dots) =$ true, where $P()$ is some polynomial time program. Clearly, this is quite a powerful cryptographic tool.

3.4 Applying ZK to Palladium

The Secure Support Component (SSC) of Palladium is a hardware module that can perform certain cryptographic operations as well as securely store one or more cryptographic keys. It has the public-private key pair (SK_0, PK_0) that is burned into the machine (as we mentioned earlier).

I want to convince a CA in zero-knowledge that I know SK_0, PK_0, C_0, SK_1 such that:

- SK_0 is the secret key for PK_0
- C_0 is a certificate from Dell on PK_0
- SK_1 is the secret key for PK_1 (CA knows PK_1)

At this point if the CA is convinced it returns $cert(PK_1)$. Thus the CA can certify PK_1 *without* knowing how to link it to the original public key PK burnt into the SSC by the manufacturer.