YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Notes 19 (rev. 2)*
Professor M. J. Fischer *November 14, 2006*

Lecture Notes 19

93 Composing Zero-Knowledge Proofs

One round of the simplified FFS protocol has probability 0.5 of error. That is, Mallory could fool Bob half the time. This is unacceptably high for most applications. By repeating the protocol t times, we reduce this error probability to $1/2^t$. Taking $t = 20$, for example, reduces the probability of error to less than on in a million. The downside of such *serial repetition* is that it also requires t round trip messages between Alice and Bob (plus a final message from Alice to Bob).

94 Parallel Execution of Zero-Knowledge Proofs

One could also imagine running the t executions of the protocol in parallel. Let (x_i, b_i, y_i) be the messages exchanged during the i^{th} execution of the simplified FFS protocol, $1 \leq i \leq t$. In a protocol execution, Alice sends $(x_1, \ldots, x_t)$ to Bob, then Bob sends $(b_1, \ldots, b_t)$ to Alice, then Alice sends $(y_1, \ldots, y_t)$ to Bob, and finally Bob checks the t sets of values he has received, accepting only if all checks pass.

A parallel execution is certainly attractive in practice, for it reduces the number of round-trip messages between Alice and Bob to only 1 1/2. The downside is that the resulting protocol may not be zero knowledge by our definition. Intuitively, the important difference between the serial and parallel versions of the protocol is that in the latter, Bob gets to know all of the x_i's before choosing the b_i's. While it seems implausible that this would actually help a cheating Bob to compromise Alice secret, the simulation proof used to show that a protocol is zero knowledge no longer works. First Sam would have to guess $(\hat{b}_1, \ldots \hat{b}_t)$. Then he can construct the x_i's and y_i's as before. But when he finally gets to the point in Mallory's program that Mallory generates the b_i's, the chance is very high that his initial guesses were wrong and he will be forced to start over again. Indeed, the probability that all t initial guesses are correct is only $1/2^t$.

95 Full Feige-Fiat-Shamir Authentication Protocol

The full Feige-Fiat-Shamir Authentication Protocol combines ideas of serial and parallel execution to get a protocol that exhibits some of the properties of both.

A *Blum prime* is a prime p such that $p \equiv 3 \pmod 4$. A *Blum integer* is a number $n = pq$, were p and q are both Blum primes. A special property of Blum integers is that if a is a quadratic residue modulo n, then exactly one of a's four square roots modulo n is itself a quadratic residue.

To see this, note that a is a quadratic residue modulo both p and q. Claim 3, section 64, of lecture notes 12, shows that one of a's two square roots is a quadratic residue modulo p, say b_p. Then $-b_p$ is not a quadratic residue since -1 is not a quadratic residue modulo the Blum prime p by the Euler Criterion of lecture notes 12, section 63. Similar, exactly one of a's two square roots modulo q is a quadratic residue. Applying the Chinese Remainder Theorem, it follows that exactly one of a's four square roots modulo n is a quadratic residue.

To generate the public and private keys of the full FFS protocol, Alice chooses a Blum integer n. Next she chooses random numbers $s_1, \ldots, s_k \in \mathbf{Z}_n^*$ and random bits $c_1, \ldots, c_k \in \{0, 1\}$. From these, she computes $v_i = (-1)^{c_i} s_i^{-2} \bmod n$, for $i = 1, \ldots, k$. She makes $(n, v_1, \ldots, v_k)$ public and keeps $(n, s_1, \ldots, s_k)$ private.

Notice that every v_i is either a quadratic residue or the negation of a quadratic residue. It is easily shown that all of the v_i have Jacobi symbol 1 modulo n. A round of the protocol itself is shown in Figure 1. The protocol is repeated for a total of t rounds.

	Alice		Bob
1.	Choose random $r \in \mathbf{Z}_n - \{0\}$. Choose random $c \in \{0, 1\}$. Compute $x = (-1)^c r^2 \bmod n$	$\xrightarrow{x}$	
2.		$\xleftarrow{b_1, \ldots, b_k}$.	Choose random $b_1, \ldots, b_k \in \{0, 1\}$.
3.	Compute $y = r s_1^{b_1} \cdots s_k^{b_k} \bmod n$.	$\xrightarrow{y}$	
4.			Compute $z = y^2 v_1^{b_1} \cdots v_k^{b_k} \bmod n$. Check $z \equiv \pm x \pmod{n}$ and $z \neq 0$.

Figure 1: One round of the full Feige-Fiat-Shamir authentication protocol.

To see that the protocol works when both Alice and Bob are honest, one needs to verify that Bob's checks will succeed. Plugging in for y, we get that

$$z = r^2 (s_1^{2b_1} \cdots s_k^{2b_k})(v_1^{b_1} \cdots v_k^{b_k}) \bmod n.$$

Since $v_i = (-1)^{c_i} s_k^{-2}$, it follows that $s_i^2 v_i = (-1)^{c_i}$. Hence,

$$z \equiv r^2 (s_1^2 v_1)^{b_1} \cdots (s_k^2 v_k)^{b_k} \equiv x(-1)^c (-1)^{c_1 b_1} \cdots (-1)^{c_k b_k} \equiv \pm x \pmod{n}.$$

Moreover, since $x \neq 0$, then also $z \neq 0$. Hence, Bob's checks succeed.

The chance that a bad Alice can fool Bob is only $1/2^{kt}$. The authors recommend $k = 5$ and $t = 4$ for a failure probability of $1/2^{20}$.

96 Non-interactive Interactive Proofs

We have seen that going from a serial composition of interactive proofs to a parallel version reduces communication overhead but possibly at the sacrifice of zero knowledge. Rather surprisingly, one can go a step further to eliminate the interaction from interactive proofs altogether.

The idea here is that Alice will provide Bob with a trace of an execution of herself in an interactive protocol with Bob. Bob will check the trace to make sure that it is a possible record of an interaction between the two of them when both are following the protocol. Of course, that isn't enough to convince Bob that Alice isn't cheating, for how does he ensure that Alice simulates random query bits b_i for him, and how does he ensure that Alice chooses her x_i's before knowing the b_i's?

The solution to both of these puzzles is to make the b_i's depend in an unpredictable way on the x_i's. We do this by choosing the b_i's from the value of a hash function applied to the concatenation of the x_i's. Here's how it works in, say, the parallel composition of t copies of the simplified FFS protocol. The honest Alice chooses $x_1, \ldots, x_t$ according to the protocol. Next she chooses

$b_1 \dots b_t$ to be the first t bits of $H(x_1 \cdots x_t)$. Finally, she computes $y_1, \dots, y_t$, again according to the protocol. She sends Bob a single message consisting of $x_1, \dots, x_t, y_1, \dots, y_t$. Bob computes $b_1 \dots b_t$ to be the first t bits of $H(x_1 \cdots x_t)$ and then performs each of the t checks of the FFT protocol, accepting Alice's proof only if all checks succeed.

A cheating Alice can choose y_i arbitrarily and then compute a valid x_i for a given b_i. But if she chooses the b_i's first, the chance that the x_i's she then computes will hash to a string that begins with $b_1 \dots b_t$ is only $1/2^t$.[1] If some b_i does not agree with the corresponding bit of the hash function, she can either change b_i and try to find a new y_i that works with the given x_i, or she can change x_i to try to get the i^{th} bit of the hash value to change. However, neither of these approaches works. The former requires knowledge of Alice's secret; the latter will cause all of the bits of the hash function to change "randomly".

Note that one way Alice can attempt to cheat is to use a brute-force attack. For example, she could generate all of the x_i's to be squares of the y_i with the hopes that the hash of the x_i's will make all $b_i = 0$. But that is likely to require 2^{t-1} attempts on average. If t is chosen large enough (say $t = 80$), the number of trials Alice would have to do in order to have a significant probability of success is prohibitive.

Of course, these observations are not a proof that Alice can't cheat; only that the obvious strategies don't work. Nevertheless, it is plausible that a cheating Alice not knowing Alice's secret, really wouldn't be able to find a valid such "non-interactive interactive proof".

Even if this is so, there is an important difference between the true interactive proofs we have been discussing and this kind of non-interactive proof. With a true zero-knowledge interactive proof, Bob does not learn anything about Alice's secret, nor can Bob impersonate Alice to Carol after Alice has authenticated herself to Bob. On the other hand, if Alice sends Bob a valid non-interactive proof, then Bob can in turn send it on to Carol. Even though Bob couldn't have produced it on his own, it is still valid. So here we have the curious situation that Alice needs her secret in order to produce the non-interactive proof string π, and Bob can't learn Alice's secret from π, but now Bob can use π itself in an attempt to impersonate Alice to Carol.

97 Feige-Fiat-Shamir Signatures

A signature scheme has a lot in common with the "non-interactive interactive" proofs introduced in section 96. In both cases, there is only a one-way communication from Alice to Bob. Alice signs a message and sends it to Bob. Bob then verifies it without further interaction with Alice. If Bob hands the message to Carol, then Carol can also verify that it was signed by Alice.

Not surprisingly, the "non-interactive interactive proof" ideas can be used to turn the Feige-Fiat-Shamir authentication protocol of section 95 into a signature scheme. The signature scheme we present here is based on a slightly simplified version of the aforementioned protocol in which all of the v_i's in the public key are quadratic residues, and n is not required to be a Blum integer, only a product of two distinct odd primes. The public verification key is $(n, v_1, \dots, v_k)$, and the private signing key is $(n, s_1, \dots, s_k)$, where $v_j = s_j^{-2} \bmod n$ $(1 \le j \le k)$.

To sign a message m, Alice simulates t rounds of the protocol in parallel. She first chooses random $r_1, \dots, r_t \in \mathbf{Z}_n - \{0\}$ and computes

$$x_i = r_i^2 \bmod n \; (1 \le i \le t).$$

[1]This assumes that the hash function "looks like" a random function. We have already seen artificial examples of hash functions that do not have this property.

Next she computes $u = H(mx_1 \cdots x_t)$, where H is a suitable cryptographic hash function. She chooses $b_{1,1}, \ldots, b_{t,k}$ according to the first tk bits of u, that is,

$$b_{i,j} = u_{(i-1)*k+j} \ (1 \leq i \leq t,\ 1 \leq j \leq k).$$

Finally, she computes

$$y_i = rs_1^{b_{i,1}} \cdots s_k^{b_{i,k}} \bmod n \ (1 \leq i \leq t).$$

The signature is

$$s = (b_{1,1}, \ldots, b_{t,k}, y_1, \ldots, y_t).$$

To verify the signed message (m, s), Bob computes

$$z_i = y_i^2 v_1^{b_{i,1}} \cdots v_k^{b_{i,k}} \bmod n \ (1 \leq i \leq t).$$

Bob checks that each $z_i \neq 0$ and that $b_{1,1}, \ldots, b_{t,k}$ are equal to the first tk bits of $H(mz_1 \cdots z_t)$.

When both Alice and Bob are honest, it is easily verified that $z_i = x_i$ $(1 \leq i \leq t)$. In that case, Bob's checks all succeed since $x_i \neq 0$ and $H(mz_1 \cdots z_t) = H(mx_1 \cdots x_t)$.

To forge Alice's signature, an impostor must find $b_{i,j}$'s and y_i's that satisfy the equation

$$b_{1,1} \ldots b_{t,k} \preceq H(m(y_1^2 v_1^{b_{1,1}} \cdots v_k^{b_{1,k}} \bmod n) \ldots (y_t^2 v_1^{b_{t,1}} \cdots v_k^{b_{t,k}} \bmod n)).$$

where "$\preceq$" means string prefix. It is not obvious how to solve such an equation without knowing a square root of each of the v_i^{-1}'s and following essentially Alice's procedure.

98 Two-Part Secret Splitting

There are many situations in which one wants to grant access to a resource only if a sufficiently large group of agents cooperate. For example, a store safe might require both the manager's key and the armored car driver's key in order to be opened. This protects the store against a dishonest manager or armored car driver, and it also prevents an armed robber from coercing the manager into opening the safe. A similar 2-key system is used for safe deposit boxes in banks.

We might like to achieve the same properties for cryptographic keys or other secrets. For example, if k is the secret decryption key for a cryptosystem, one might wish to split k into two *shares* k_1 and k_2. By themselves, neither k_1 nor k_2 reveals any information about k, but when suitably combined, k can be recovered. A simple way to do this is to choose k_1 uniformly at random and then let $k_2 = k \oplus k_1$. Both k_1 and k_2 are uniformly distributed over the key space and hence give no information about k. However, combined with XOR, they reveal k, since $k = k_1 \oplus k_2$.

Indeed, the one-time pad cryptosystem of lecture notes 3, section 14, can be viewed as an instance of secret splitting. Here, Alice's secret is her message m. The two shares are the ciphertext c and the key k. Neither by themselves gives any information about m, but together they reveal $m = k \oplus c$.

99 Multi-Part Secret Splitting

Secret splitting generalizes to more than two shares. Imagine a large company that restricts access to important company secrets to only its five top executives, say the president, vice-president, treasurer, CEO, and CIO. They don't want any executive to be able to access the data alone since they are concerned that an executive might be blackmailed into giving confidential data to a competitor.

On the other hand, they also don't want to require that all five executives get together to access their data, both because this would be cumbersome and also because they worry about the death or incapacitation of any single individual. They decide as a compromise that any three of them should be able to access the secret data, but not one or two of them operating alone.

A (τ, k) *threshold secret splitting scheme* splits a secret s into shares $s_1, \ldots, s_k$. Any subset of τ or more shares allows s to be recovered, but no subset of shares of size less than τ gives any information about s.