MidTerm Exam Solution

CSCI 556: Introduction to Cryptography

Non-Interactive Zero Knowledge Proof Steps:

Peggy knows discrete logarithm of y to the base g, $y = g^x \bmod p$

1. Peggy chooses random k, computes $t = g^k \bmod p$
2. Peggy computes hash value, $h = H(y, t)$
3. Peggy computes $r = k - hx \bmod p - 1$ and sends (t, h, r) to Victor
4. Victor verifies $t = g^r y^h \bmod p$ and $h = H(y, t)$

Question 1. $h = H(y), t = g^r y^h \bmod p$

Answer: As a forger, Peggy can find a triple of values (t, h, r) and possibly y that will fool the verifier Victor without using $x = \log_g y$ directly.

x = secret key, $x = \log_g y$

$y = g^x \bmod p$

For example: Let's say Peggy chooses a random r_1 value instead of actual r value. Now, using this r_1 value Peggy computes t_1 value using the equation

$$t_1 = g^{r1} y^h \bmod p$$

Now, Peggy sends this (t_1, h, r_1) value to Victor and Victor verifies them again using the same above equation ($t_1 = g^{r1}y^h \bmod p$). So, here clearly Peggy has cheated Victor. Hence, the forger already knows the required values to fool the verifier if $h = H(y)$

The Fiat-Shamir heuristic hash function doesn't include any random factor. Hence the challenge generated by this cryptographic hash function is not random either as it only depends on y every time. So, it is important to hash prover's first message key while generating random value in step [2].

Question 2. $h = H(t), t = g^r y^h \bmod p$

Answer:

The main idea behind Fiat-Shamir heuristic is to replace the interaction in step [2] with some random oracle model.

The randomly selected hash values from this random oracle model or cryptographic hash function force a malicious prover to provide a wrong proof. It is very negligible for the malicious prover to provide correct values or to prove herself to the verifier after k rounds.

To make this possible, the cryptographic hash function used, or the Fiat-Shamir heuristic must produce an output that is not distinguishable from a truly random value. This value must be unpredictable.

It means the prover is not able to predict the output of this hash function in each iteration round of the verification. Which means the prover must commit to y and t before reading this challenge or hash value.

If prover doesn't commit to y or t before reading hash value, then prover has the chance to do forgery i.e Peggy can convince to Victor that she knows discrete logarithm without actually knowing it.

$$h = H(t) \qquad t = g^r y^h \bmod p$$
$$y^h = t / g^r = d \qquad \text{consider } d = t / g^r$$

Let's say Peggy picks t and computes h. Now we already know g, t and h. With the range of r values, Peggy chooses a random r value that can solve the above equation.

$$y^h = d \Rightarrow \text{Using Lemma problem, we can consider } y = d^s \bmod p \text{ where } h*s = 1 \bmod p\text{-}1$$
$$(d^s)^h = d^{sh} = d^{1+q\,(p-1)} = d * (d^{p-1})^q = d * 1^q = d \bmod p$$

Using Fermat's little theorem, we computed y given (t, h, r) values. Which means we can compute or modify y to fit the above formula since the hash value h doesn't depend on y. Now, Peggy can cheat by sending values (t, h, r) which satisfies the above equation without committing to y in the hash function.

As we can see in both the cases, Peggy can cheat Victor. So, it is necessary for Peggy to commit to both y and t in the hash function to avoid this kind of forgery or cheating.

Question 3. Digital Signature Scheme

Answer:

All users of the signature scheme agree on a group G, of prime order p, with generator g, in which the discrete log problem is assumed to be hard. All users agree on a cryptographic hash function H. Hence, the public parameters consist of a prime p and generator g for the group Z_p^*

Key Generation Phase:

1. Choose a private signing key x from the allowed set of multiplicative group of integers modulo p.
2. The public verification key is then computed as $y = g^x = g^x \bmod p$

Signature Phase:

1. Peggy chooses a random value k from the allowed set of multiplicative group of integers modulo p i.e Z_{p-1}.
2. Peggy computes public key, $t = g^k = g^k \bmod p$
3. Peggy computes the challenge, h = H(m, y, t) where H is the cryptographic collision resistant hash function (so called oracle model), m is the message to be signed, y is the public verification key and t is the first message key generated previously. y and t are included as parameters in the hash function to prevent Peggy from cheating.
4. Peggy computes $r = k - hx \bmod p\text{-}1$
5. Peggy sends $\sigma = (t, h, r)$ to Victor as the signature pair

Verification Phase:

1. Victor receives the signature pair from Peggy, $\sigma = (t, h, r)$
2. Victor computes $t_1 = g^r y^h \bmod p$
3. Victor computes $h_1 = H(m, y, t_1)$
4. If $h_1 = h$, then the signature is verified

Proof of Correctness:

1. It is relatively easy to see $h_1 = h$ if the signed message equals the verified message.
2. $t_1 = g^r y^h = g^{k-hx} g^{hx} = g^k = t$
3. Hence $h_1 = H(m, y, t_1) = H(m, y, t) = h$
4. This shows that a correctly signed message will verify correctly.

Security Considerations:
1. A digital signature scheme is secure if the discrete logarithm problem is intractable. The following properties are satisfied by the non-interactive zero knowledge proof.
2. Completeness: a prover who knows discrete logarithm always passes the verification.
3. Soundness: adversary who doesn't know discrete logarithm has a very negligible probability (2^{-t}) to pass the verification.
4. Zero-Knowledge: prover doesn't share any other information to the verifier, regardless of whether the prover knows discrete logarithm.

Collision Resistant Hash function:

A cryptographic hash function is said to be collision resistant if it is hard to find two inputs that hash to the same output, that is two inputs a and b such that $a \neq b$ and $H(a) = H(b)$.

Collision resistant hash functions will have output space larger than input space. If not, there are chances of collisions. Since the collision resistant hash function depends on both y and t in the non-interactive zero knowledge proof we discussed earlier, the prover can't cheat the verifier. The prover has to commit to both y and t before getting the random challenge value from this hash function. In that way, the prover cannot alter either y or t later this step preventing forgery.

The collision resistance property produces true random value from the hash function. Hence it prevents chosen plaintext and chosen message attack. This collision resistant cryptographic hash function acts like a random oracle model in this scenario. So, heuristically speaking it generates random value and prevents any forgery providing the necessary security for the signature scheme.

Question 4. Example/Application of Non-Interactive Zero Knowledge (NIZK) Proof

Answer: **J-PAKE: Password Authenticated Key Exchange by Juggling**

Password-Authenticated Key Exchange (PAKE) is a technique that aims to establish secure communication between two remote parties solely based on their shared password, without relying on a Public Key Infrastructure or any trusted third party. J-PAKE is a modified version of PAKE designed by Feng Hao and Peter Ryan in 2008.

There are a few factors that may be considered in favor of J-PAKE. First, J-PAKE has security proofs, while equivalent proofs are lacking in similar protocols. Second, J-PAKE follows a completely different design approach from all other PAKE protocols and is built upon a well-established Zero Knowledge Proof (ZKP) primitive: Schnorr NIZK proof. Third, J-PAKE optimize the use of ZKP so that overall the protocol is sufficiently efficient for practical use. Finally, J-PAKE has been used in real-world applications at a relatively large scale, e.g., Firefox sync, Pale moon sync and Google Nest products. It has been included into widely distributed open source libraries such as OpenSSL, Network Security Services (NSS), and the Bouncy Castle. Since 2015, J-PAKE has been included in Thread as a standard key agreement mechanism for IoT (Internet of Things) applications, and also included in ISO/IEC 11770-4:2017.

Protocol SetUp:
Let p and q be two large primes such that q | p-1. Let G_q denote a subgroup of Zp* with prime order q. Let g be a generator for G_q. The two communicating parties, Alice (Prover) and Bob (Verifier), both agree on (p, q, g). Let s be a secret value derived from a low-entropy password shared between Alice and Bob. The value of s is REQUIRED to fall within the range of [1, q-1]. In a practical implementation, one may obtain s by taking a cryptographic hash of the password and wrapping the result with respect to modulo q.

Two-Round Key Exchange:

Round 1: Alice selects an ephemeral private key x1 uniformly at random from [0, q-1] and another ephemeral private key x2 uniformly at random from [1, q-1]. Similarly, Bob selects an ephemeral private key x3 uniformly at random from [0, q-1] and another ephemeral private key x4 uniformly at random from [1, q-1].

- Alice $\Rightarrow$ Bob: g1 = g^{x1} mod p, g2 = g^{x2} mod p and ZKPs for x1 and x2
- Bob $\Rightarrow$ Alice: g3 = g^{x3} mod p, g4 = g^{x4} mod p and ZKPs for x3 and x4

In this round, the sender must send zero knowledge proofs to demonstrate the knowledge of the ephemeral private keys. A suitable technique is to use the NIZK proof such as Schnorr NIZKP. As an example, suppose one wishes to prove the knowledge of the exponent for y = g^x mod p. The generated NIZK proof will contain: {UserID, t = g^k mod p, r = k - hx mod q}, where UserID is the unique identifier for the prover, v is a number chosen uniformly at random from [0, q-1] and h = H(g || t || y || UserID). When this round finishes, Alice verifies the received ZKPs and checks g4 != 1 mod p. Similarly, Bob verifies the received ZKPs and checks that g2 != 1 mod p.

Round 2: In this round, the Schnorr NIZK proof is computed in the same way as in the previous round except that the generator is different.

- Alice $\Rightarrow$ Bob: A = $(g1*g3*g4)^{(x2*s)}$ mod p and a ZKP for x2*s
- Bob $\Rightarrow$ Alice: B = $(g1*g2*g3)^{(x4*s)}$ mod p and a ZKP for x4*s

Alice and Bob just need to ensure g1*g3*g4 != 1 mod p and g1*g2*g3 != 1 mod p. The receiving party shall explicitly check if these inequalities hold and abort the session in case such a check fails. When the second round finishes, Alice and Bob verify the received ZKPs. If the verification fails, the session is aborted. Otherwise, the two parties compute the common key material as follows:

- Alice computes K_a = $(B/g4^\wedge(x2*s))^{x2}$ mod p
- Bob computes K_b = $(A/g2^\wedge(x4*s))^{x4}$ mod p

Here, K_a = K_b = g^((x1+x3)*x2*x4*s) mod p. Let K denote the same key material held by both parties. Using K as input, Alice and Bob then apply a Key Derivation Function (KDF) to derive a common session key k.

Computational Cost: Alice takes 14 modular exponentiations in total. Based on the symmetry, the computational cost for Bob is exactly the same.

Key Confirmation:
The two-round J-PAKE protocol provides cryptographic guarantee that only the authenticated party who used the same password at the other end is able to compute the same session key. The two parties may use the derived key straight away to start secure communication by encrypting messages in an authenticated mode. Only the party with the same derived session key will be able to decrypt and read those messages. So far, the authentication is only implicit. For achieving explicit authentication, an additional key confirmation procedure should be performed.

Security considerations:

1. Offline dictionary attack resistance: It does not leak any information that allows a passive/active attacker to perform offline exhaustive search of the password.
2. Forward secrecy: It produces session keys that remain secure even when the password is later disclosed.
3. Known-key security: It prevents a disclosed session key from affecting the security of other sessions.
4. Online dictionary attack resistance: It limits an active attacker to test only one password per protocol execution.

References:

1. Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In Advances in Cryptology - CRYPTO ’86, Vol. 263 of Lecture Notes in Computer Science: pp. 186-194, Springer (1986).
2. Hao, F. and P. Ryan, "Password Authenticated Key Exchange by Juggling", Lecture Notes in Computer Science, pp. 159-171, from 16th Security Protocols Workshop (SPW'08).
3. Hao, F., Ed., "Schnorr Non-interactive Zero Knowledge Proof", DOI 10.17487/RFC8235, September 2017.
4. David Bernhard, Olivier Pereira, Bogdan Warinschi, "How not to prove yourself: pitfalls of the fiat-shamir heuristic and applications to helios", Proceedings of the 18th international conference on The Theory and Application of Cryptology and Information Security, December 02-06, 2012, Beijing, China.
5. David, "Schnorr's Signature and non-interactive Protocols", cryptogie.net, December 2014.
6. Suprabha Shreepad Hegde, "Analysis of Non-Interactive Zero-Knowledge Proof", MS Thesis Dissertation, Dept of Computer Science, University of Cincinnati, July 2018.
7. "Schnorr Signature", Wikipedia, Accessed on October 20, 2019.
8. "Fiat-Shamir Heuristic", Wikipedia, Accessed on October 19, 2019.
9. Dan Boneh, Victor Shoup, "A Graduate Course in Applied Cryptography", Version 0.4, September 2017.
10. Jonathan Katz, Yehudu Lindell, "Introduction to Modern Cryptography", Second Edition, CRC Press, A Chapman & Hall Book, 2015.