A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup

Chae Hoon Lim and Pil Joong Lee*

POSTECH Information Research Laboratories
Pohang University of Science and Technology (POSTECH)
Pohang, 790-784, KOREA
E-mail: chlim@oberon.postech.ac.kr; pjl@postech.ac.kr

February 6, 1997

Abstract

Consider the well-known oracle attack: Somehow one gets a certain computation result as a function of a secret key from the secret key owner and tries to extract some information on the secret key. This attacking scenario is well understood in the cryptographic community. However, there are many protocols based on the discrete logarithm problem that turn out to leak many of the secret key bits from this oracle attack, unless suitable checkings are carried out. In this paper we present a key recovery attack on various discrete log-based schemes working in a prime order subgroup. Our attack can disclose part of, or the whole secret key in most Diffie-Hellman-type key exchange protocols and some applications of ElGamal encryption and signature schemes.

Key Words : Key recovery attack, Discrete logarithms, Key exchange, Digital signatures.

1 Introduction

Many cryptographic protocols have been developed based on the discrete logarithm problem. The main objective of developers is to design a protocol that is as difficult to break as the underlying discrete logarithm problem under some reasonable assumptions. On the other hand, the goal of attackers is to find a way to extract the secret key involved or to pretend to be a legitimate user without knowing the secret key. Though provable security guarantees that there is no efficient attack on the protocol, it should be carefully interpreted in practical implementations (e.g., one should consider the assumptions on which the security proof is based). The most fundamental is to use secure parameters and check the properties or requirements assumed to be satisfied. To see this, for example, we refer the reader to Pointcheval and Stern [33] for security proof and Bleichenbacher [4] for signature forgery in ElGamal-type signature schemes, together with Stern [38] for further discussions on their apparent contradiction.

The purpose of this paper is to reveal the insecurity of various protocols based on the discrete logarithm problem published in the literature. We present a key recovery attack on these protocols

*This work was done while the second author was in the NEC Research Institute, NJ, Princeton, during his sabbatical leave.

which can find substantial part of the secret key involved. Our attack only works for discrete log-based schemes using a prime order subgroup and is closely related to the choice of parameters and the checking of protocol variables. Thus, as is usual, our attack, once identified, can be easily prevented by adding suitable checking steps or by using 'secure' parameters. Here 'secure' means that the parameters are secure against our attack. And this also implies that the usual parameters commonly used in the literature are not secure against our attack. Our attack demonstrates the importance of checking protocol variables in designing discrete log-based schemes: we will show that our attack can be applied even to a protocol known to be zero-knowledge, unless a new checking step is added.

Symbols and Notation : The following symbols and notation will be used throughout this paper, unless otherwise stated: A prime p is assumed to be chosen at random so that $p-1$ has a large prime factor q (usually, $|p| > 512$ and $|q| = 160$) and thus $(p-1)/q$ may have small prime factors. Let g be a base element of order q mod p (i.e., $g \neq 1$ and $g^q = 1 \bmod p$) and β be an element of small order mod p, where 'small' means that it is small compared to q (usually between 0 and 2^{80} for the feasibility of our attack, depending on specific instances). Let $\gamma = \prod \beta_i$, a product of small order elements with β_i taken from each small order subgroup. The order of an element β mod p is denoted by $ord(\beta)$ and a secure hash function by h. The notation $k \in_R \mathbf{Z}_q$ denotes that k is chosen at random in $\mathbf{Z}_q$. We assume that an honest user i has a secret key $x_i \in_R \mathbf{Z}_q$ and the corresponding public key $y_i = g^{x_i} \bmod p$ (sometimes we omit the subscript).

Pohlig-Hellman Decomposition and Pollard's Methods : The discrete logarithm problem over $\mathbf{Z}_p^*$ can be broken down into a number of small such sub-problems defined over small order subgroups of $\mathbf{Z}_p^*$ (Pohlig-Hellman decomposition [32]). Then these sub-problems can be solved using Pollard's rho and lambda methods [34] and the resulting partial logarithms can be combined using the Chinese Remainder Theorem to give the pursued discrete logarithm.

For simplicity, suppose that $p-1 = \prod_{i=1}^{n} q_i$ (q_i prime). Let α be a generator of $\mathbf{Z}_p^*$. Given $y = \alpha^x \bmod p$, we can reduce the problem of finding x mod $p-1$ to the following sub-problems: find $x_i = x \bmod q_i$ from $y_i = y^{(p-1)/q_i} = \alpha_i^{x_i} \bmod p$ for each i, where $\alpha_i = \alpha^{(p-1)/q_i}$ (an element of order q_i). Each such sub-problem can be solved using Pollard's rho method in time $O(\sqrt{q_i})$ (see [39] for linear speedup with multiple processors). Once x_i's are found for all i, they can be combined using the CRT, yielding the logarithm x. Pollard's rho method is the most efficient in computing a logarithm in a prime order subgroup, while Pollard's lambda method can compute a logarithm that is known to lie within some restricted interval of width w in time $O(\sqrt{w})$. Thus, both methods have similar square-root running time for a given size of an unknown exponent. In particular, the lambda method is very useful for computing a logarithm in a prime order subgroup when part of the logarithm is known. See van Oorschot and Wiener [40] for detailed explanations on the combined use of these methods, together with a Pohlig-Hellman decomposition, to speed up computing a complete logarithm.

The Attacking Scenario : In this paper we pay our attention to DL-based schemes using a prime order subgroup. Thus, as is usual, we assume that a prime p is chosen such that $p-1$ has a large prime factor q and that a base g is an element of order q. Then, for a given y such that $y = g^x \bmod p$, it is completely infeasible under current technology to find x using Pollard's rho method, if we take for example $|q| = 160$, since it requires about 2^{80} operations. Our observation is that if we could obtain $z = \gamma^x \bmod p$ somehow by attacking a protocol, where $\gamma = \prod \beta_i$ (a product of distinct smooth order elements mod p), then we could find x modulo $ord(\gamma)$ using the Pohlig-Hellman decomposition. Here we assume that $(p-1)/q$ has many small prime factors, which is the case for a randomly generated prime p. And finally the remaining part of x could be found from y using Pollard's lambda method. A special case of the attack is to find x mod $ord(\beta)$, given $z = f(\beta^x \bmod p)$ for any function f. In this case, one can find $j = x \bmod ord(\beta)$ by checking that $z = f(\beta^j \bmod p)$ for $j = 0, 1, \cdots, ord(\beta) - 1$.[1]

The main problem in the above attacking scenario is how to obtain a Pohlig-Hellman decomposition for the secret key. This should be impossible in well-designed protocols. However, we could find many DL-based schemes susceptible to the above attack in the literature. Most Diffie-Hellman type key exchange protocols are vulnerable to the above attack. Other examples include shared decryption of ElGamal encryption, shared verification of ElGamal signatures and undeniable signatures. Our attack was possible in all these schemes, since the involved parties do not check relevant protocol variables. Though there are several papers pointing out the importance of checking public parameters and protocol variables (e.g., see [4, 41, 1, 40, 2]) in DH key exchange and digital signature schemes, no literature addresses such an explicit attack revealing the involved secret. Our attack may find the whole secret key in many cases.

Related Work : Previous work most relevant to our attack is the middleperson attack on the original Diffie-Hellman key exchange protocol [16] (see [40, 2]). Two parties A and B agree on a prime p and a generator α of $\mathbf{Z}_p^*$, exchange random exponentials, $r_A = \alpha^{k_A} \bmod p$ and $r_B = \alpha^{k_B}$ mod p, and then compute a shared secret $K = r_B^{k_A} = r_A^{k_B} = \alpha^{k_A k_B} \bmod p$. Suppose that $p-1 = qw$ with w smooth. An attacker may replace r_A and r_B with $r_A^q \bmod p$ and $r_B^q \bmod p$ respectively. Then the shared key becomes $K = (\alpha^q)^{k_A k_B} \bmod p$, which can also be computed by the attacker since he can find k_i mod w from r_i. This attack can be easily prevented by authenticating the random exchange, as in the STS [17] and SKEME [21] protocols.[2]

The above attack motivates the use of a prime order subgroup, which also substantially increases the efficiency in computation and parameter generation (see [40, 2] for further discussions). Thus most DL-based schemes have been designed using a prime order subgroup since its first invention by Schnorr [37]. However, this paper will show potential weaknesses in such a setting. Our attack on key exchange protocols is quite similar to the above attack, except that our target protocols use a prime order subgroup and that our objective is to find the long-term secret key of the involved party (usually by the other legitimate party). Our attack can be applied to any protocol involving a DH shared secret.

The rest of this paper is organized as follows: We present in Sec.2 a key recovery attack on DH-type key exchange protocols and in Sec.3 a similar attack on other DL-based schemes such as ElGamal encryption and signatures. Sec.4 deals with the generation of secure primes and public key certificates as possible countermeasures to minimize security loss by our attack. And we conclude in Sec.5.

2 Extracting Secret Keys in Key Exchange Protocols

One of the well-known design principles for public key protocols states that a message received should not be assumed to have a particular form unless it can be checked [1]. In particular, it is very dangerous to apply one's secret to a number received from the other. However, this principle

[1]Here we assumed that the range of f is large enough compared to β, so that the probability of collisions occurring under f is negligible. This is usually the case in most instances of our attack.

[2]It is very important to authenticate the exchanged random messages themselves, rather than the shared secret computed from them. For example, the modified STS protocol by Boyd and Mao [5] may be vulnerable to the middleperson attack, since it only authenticates the hashed version of the shared secret.

is hard to apply to Diffie-Hellman-type key exchange protocols. This has given rise to a lot of attacks or weaknesses under a variety of attacking scenarios. Most attacks aim at finding a session key (e.g., see [10, 42]) or causing authentication failure (e.g., see [27]). In this section we present a key recovery attack that can be applied to many DH-type key exchange protocols published in the literature[3] unless proper precautions are taken additionally.

2.1 Basic Diffie-Hellman Key Exchange

We first consider the case where a user A successfully obtained a certificate on the public key $y_A = \beta g^{x_A} \bmod p$ with β of small order mod p. This is possible unless a certification authority checks that $y_A^q = 1 \bmod p$ before issuing a certificate for y_A. The CA usually requires that each user prove knowledge of a secret key corresponding to the public key to be certified, since otherwise there exist some protocols that can be attacked with a faked public key (e.g., see [27]). However, even in this case it may still be possible to register a public key of the form $y_A = \beta g^{x_A} \bmod p$ if it is not checked that $y_A^q = 1 \bmod p$.

For example, suppose that for registration the CA requires a user's digital signature on the certificate message which contains all necessary information for certification, including the public key, as defined by X.509. In this case it is easy for A to generate a valid signature corresponding to the public key $y_A = \beta g^{x_A} \bmod p$ when $ord(\beta)$ is small. For example, suppose that Schnorr's signature scheme [37] is used for this purpose. Given message m, A can find $r' \in (0, ord(\beta)]$ such that $r' = h(\beta^{r'} g^k \bmod p,\ m) \bmod ord(\beta)$ in about $ord(\beta)$ steps, where $k \in_R \mathbf{Z}_q$ and h denotes a secure hash function. Thus A can generate a signature $\{r, s\}$ on m by computing $r = h(\beta^{r'} g^k \bmod p,\ m)$ and $s = k - xr \bmod q$. It is easy to see that the resulting $\{r, s\}$ is a valid signature on m with the public key $y_A = \beta g^{x_A} \bmod p$. On the other hand, suppose that Schnorr's identification scheme is used instead. Then A can pass the protocol with probability $1/ord(\beta)$ on average, irrespective of the size of a challenge by B (A similar observation has been made before by Burmester [9]). Therefore, it is essential that the CA should first check that $y_A^q = 1 \bmod p$.

We now present a key recovery attack under the assumption that an attacking user i has a public key $y_i = \beta g^{x_i} \bmod p$. This attack will demonstrate the importance of the checking step in the certification process. We first consider the zero-message DH key exchange with public keys (e.g., used in [22])[4] : Two users A and B share a session key K by computing $K = h(y_B^{x_A} \bmod p, d) = h(y_A^{x_B} \bmod p, d)$, where d is time/date information. In this protocol, suppose that user B with public key $y_B = g^{x_B} \bmod p$ uses a session key computed by $K = h(y_A^{x_B} \bmod p, d)$ to send a message m to user A with public key $y_A = \beta g^{x_A} \bmod p$. Then, when receiving $\{c = E_K(m), d\}$ from B, A can extract $|ord(\beta)|$ bits of B's secret key by an exhaustive search. For this, A computes $K_j = h(y_B^{x_A} \cdot \beta^j \bmod p, d)$ for $0 \le j < ord(\beta)$, decrypts c with each K_j and checks that the result is a meaningful message. If a meaningful message is found for some j, then A has found $j = x_B \bmod ord(\beta)$.

User A may repeat this attack by updating his public key with β of a different order and combine the resulting partial secrets using the Chinese Remainder Theorem. This will give about

[3]Diffie-Hellman-type key exchange protocols can be divided into two broad classes. The first is to exchange random exponentials and then authenticate the exchange using a separate authentication mechanism. The STS and SKEME protocols belong to this class. Such protocols seem to be the most robust against various attacks, including our one. Most other protocols involve the fixed secret/public key pair for key exchange and (possibly) authentication. Our attack can be applied to most of such protocols.

[4]The SKIP protocol [3] being widely implemented in the industry also employs this scheme to get a long-term shared secret, which is used as a key-encrypting key. However, the SKIP documentation recommands to use a safe prime p, i.e., a prime p such that $(p-1)/2$ is also prime. Thus our attack on this protocol only discloses one bit of the secret, the parity bit.

t bits of x_B if t is the bit-length of small prime factors of $p-1$ that can be used for this attack. Now the remaining $(|q|-t)$ bits of x_B can be found in about $2^{(|q|-t)/2}$ steps using Shanks' method or Pollard's lambda method (see [40] for further discussions). Note that if $ord(\beta)$ is small, say 20 bits, A has little difficulty in reading the ciphertext directed to him. Also note that this attack can be applied to any protocol if the protocol reveals an equation involving the fixed DH key $g^{x_A x_B} \bmod p$.

As another example, let us consider the following protocol (modified from Protocol 3 in [23]), where we assume that B has a public key $y_B = \beta g^{x_B} \bmod p$.

1. A computes $r_A = g^{k_A} \bmod p$ with $k_A \in_R \mathbf{Z}_q$ and sends it to B.
2. B computes $r_B = g^{k_B} \bmod p$ with $k_B \in_R \mathbf{Z}_q$ and $s_B = k_B - x_B e_B \bmod q$ with $e_B = h(r_A, r_B, A, B)$. Then B sends $\{e_B, s_B\}$ to A.
3. A checks that $e_B = h(r_A, g^{s_B} y_B^{e_B} \bmod p, A, B)$. If the check succeeds, A sends $s_A = k_A - x_A e_A \bmod q$ with $e_A = h(r_B, r_A, B, A)$ to B and computes the session key $K_A = h(r_B^{k_A} \bmod p)$. Otherwise, A stops the protocol with failure.
4. B computes $e_A = h(r_B, r_A, B, A)$ and checks that $g^{s_A} y_A^{e_A} = r_A \bmod p$. If the check succeeds, B computes the session key $K_B = h(r_A^{k_B}) \bmod p$. Otherwise, B stops the protocol with failure.

Since B can generate a valid signature with $r_B = \beta^{e_B} g^{k_B} \bmod p$ in about $2^{|ord(\beta)|}$ steps as explained before, it can pass A's verification in step 3 in real time if $ord(\beta)$ is small enough (say, 10 to 20 bits). Therefore, once B obtains any checking equation involving the session key computed by A, for example a ciphertext generated by A, then it can find $k_A \bmod ord(\beta)$ by trying all possible values of $K_A = h(r_A^{k_B} \beta^j \bmod p)$ with $j \in [0, ord(\beta))$. This gives $|ord(\beta)|$ bits of information on the secret key x_A, since $x_A = (s_A + k_A) e_A^{-1} \bmod q$, and thus the effective secret bits of x_A is reduced to $(|q| - |ord(\beta)|)$ bits.

In the above protocol B has to respond in real time and thus $ord(\beta)$ must be very small. However, in the following non-interactive, symmetric protocol [27] (we only describe it w.r.t. user A only), the exhaustive search can be done in off-line, which allows to use a β of larger order (say, of 30 to 40 bits) (Of course, for this $p-1$ should have a prime factor of this size). If A has a public key $y_A = \beta g^{x_A} \bmod p$, then it can find $j = k_B \bmod ord(\beta)$ by checking that $g^{s_B} y_B^{e_B} = r_B \bmod p$ with $e_B = h(r_B^{x_A} \beta^j \bmod p, r_B, B, A)$ for all possible values of j.

1. A computes $r_A = g^{k_A} \bmod p$ with $k_A \in_R \mathbf{Z}_q$, $K_{A1} = y_B^{k_A} \bmod p$, $e_A = h(K_{A1}, r_A, A, B)$ and $s_A = k_A - x_A e_A \bmod q$. A then sends $\{r_A, s_A\}$ to B.
2. A computes $K_{A2} = r_B^{x_A} \bmod p$ and $e_B = h(K_{A2}, r_B, B, A)$, and checks that $g^{s_B} y_B^{e_B} = r_B \bmod p$. If the check succeeds, A computes the session key $K_A = h(K_{A1} K_{A2} \bmod p)$. Otherwise, A stops the protocol with failure.

The reason why our attack can be applied to the above two protocols is that the same random secret k_i is used for authentication and session key computation. This shows that a robust protocol should avoid using the same secret (even it is a one-time random number) for two different purposes [1]. In this respect the approach taken in the STS [17] and SKEME [21] seems to be a better way to design key exchange protocols.

2.2 Authenticated key Exchange

The attack presented above can be easily prevented by a proper precaution in the certificate issuing process. We now extend our attack to the case where each user has a correct public key. As an example, we consider the following key exchange protocol, which is an authenticated version of the MTI (Matsumoto-Takashima-Imai) protocol [26]. This protocol, with slight changes, is widely studied in the literature (e.g., see [27, 20]) and is also being standardized in ISO/IEC JTC1/SC27 [43].

1. A randomly picks $k_A \in \mathbf{Z}_q$, computes $r_A = g^{k_A} \bmod p$ and sends r_A to B.
2. B randomly picks $k_B \in \mathbf{Z}_q$, computes $r_B = g^{k_B} \bmod p$, $K_B = y_A^{k_B} r_A^{x_B} \bmod p$ and $e_B = h(K_B, r_B, r_A, B, A)$, and sends $\{r_B, e_B\}$ to A.
3. A computes $K_A = y_B^{k_A} r_B^{x_A} \bmod p$ and $e'_B = h(K_A, r_B, r_A, B, A)$, and checks that $e_B = e'_B$. If $e_B \neq e'_B$, then A stops the protocol with failure. (Optional) Otherwise, A computes $e_A = h(K_A, r_A, r_B, A, B)$ and sends e_A to B.
4. (Optional) B computes $e'_A = h(K_B, r_A, r_B, A, B)$ and checks that $e_A = e'_A$. If $e_A \neq e'_A$, then B stops the protocol with failure.

The session key K can be derived from the shared secret as $K = h(K_A) = h(K_B)$. The critical point relevant to our attack is the key authentication based on the shared secret $K_A = K_B$. That is, B applies his secret key to the number received from A and returns e_B as a function of the (assumed) session key K_B. Suppose that A sends $r_A = \beta g^{k_A} \bmod p$ in step 1. Then an honest user B will compute $K_B = y_A^{k_B} r_A^{x_B} = r_B^{x_A} y_B^{k_A} \beta^{x_B} \bmod p$ and return e_B computed with this K_B. Once receiving $\{r_B, e_B\}$, A may abort the protocol if a response is required. Since A can compute the first exponential in K_B, $y_A^{k_B} = r_B^{x_A} \bmod p$, it can find $j = x_B \bmod ord(\beta)$ in $O(2^{|ord(\beta)|})$ steps by checking the equality $e_B = h(K_B, r_B, r_A, B, A)$ with $K_B = r_B^{x_A} y_B^{k_A} \beta^j$ for all possible values of j (i.e., $j = 0, 1, \cdots, ord(\beta) - 1$).

The above attack may be repeated as many times as the number of small prime factors of $p-1$ which make it feasible to do the exhaustive search. Thus if $p-1$ has several prime factors of small size (say, less than 40 bits), then it is possible to find the whole secret in reasonable time. Note that our attack can be mounted against any authenticated key exchange protocol as long as authentication is performed using the shared secret (note that such authentication is possible only if each user's secret key is involved in the computation of the shared secret). This implies that almost all key exchange protocols providing explicit authentication without using a separate authentication channel (e.g., as in STS [17] or SKEME [21]) may be vulnerable to our attack.

Our attack can also be applied to key exchange protocols with implicit authentication, since the agreed upon session key will be used anyway in later communications. For example, suppose that user A mounted the attack in the original MTI protocol, where each user exchanges random exponential r_i and computes the session key as above. Now, if user B first uses the resulting session key for message authentication (or key authentication), A obtains a known equation involving the session key computed by B. Then the situation, in view of our attack, is the same as in the above authenticated protocol. On the other hand, if B sends a ciphertext for an unknown message, then A can find the intended partial secret by decrypting the ciphertext with all possible values of the session key that B is supposed to compute and then finding a meaningful message. Note that usual known-key attacks assume knowledge of the whole shared secret from which the session key is derived (e.g., see [42, 10]), but in our attack it is sufficient to obtain any function of the shared secret (even a ciphertext suffices).

We next show that some key exchange protocols using a signature scheme for authentication may also be vulnerable to our attack. For example, consider the following protocol (developed from Protocol 4 in [23]):

1. A picks a random integer $k_A \in \mathbf{Z}_q$, computes $r_A = g^{k_A} \bmod p$ and sends r_A to B.
2. B picks a random integer $k_B \in \mathbf{Z}_q$, computes $r_B = g^{k_B} \bmod p$. B also computes $K_B = r_A^{k_B} \bmod p$, $e_B = h(K_B, r_B, r_A, B, A)$ and $s_B = k_B - x_B e_B \bmod q$, and sends $\{r_B, s_B\}$ to A.
3. A computes $K_A = r_B^{k_A} \bmod p$, $e_B = h(K_A, r_B, r_A, B, A)$ and checks that $g^{s_B} y_B^{e_B} = r_B \bmod p$. If the check fails, then A stops the protocol with failure. (Optional) Otherwise, A computes $e_A = h(K_A, r_A, r_B, A, B)$ and $s_A = k_A - x_A e_A \bmod q$, and sends s_A to B.
4. (Optional) B computes $e_A = h(K_B, r_A, r_B, A, B)$ and checks that $g^{s_A} y_A^{e_A} = r_A \bmod p$. If it does not hold, then B stops the protocol with failure.

This protocol uses a digital signature on the shared secret $K_A = K_B = g^{k_A k_B} \bmod p$ (for a honest run) to authenticate each other. However, the same random number is used for authentication and session key computation (as in the last two examples in Sec.2.1). This fact can be exploited by A to extract partial information on the secret key x_B. As before, A sends $r_A = \beta g^{k_A} \bmod p$ and does the exhaustive search for $k_B \bmod ord(\beta)$ using the verification equation $g^{s_B} y_B^{h(K_A, r_B, r_A, B, A)} = r_B \bmod p$ with $K_A = r_B^{k_A} \beta^{k_B} \bmod p$. This reduces the effective secret bits of x_B to $(|q| - |ord(\beta)|)$ bits. Note, however, that repetition of the attack with β of a different order does not help to find further bits of the secret in this case, since a different k_B is used each time and $ord(\beta)$ does not divide q.

The attack described in this section can be easily prevented by checking that $r_i^q = 1 \bmod p$ for each random exponential exchanged before raising it to the secret key. This however seems too expensive. A better solution would be to choose a prime p such that $(p-1)/2q$ has prime factors that are at least larger than q (see Sec.4). Such a p only leaks the parity bit of the secret key by our attack. Note that no key exchange protocol can protect the parity bit of the involved secret if the order of the received number is not checked as explained above, since there always exist an element of order 2 (i.e., p-1). This is also true for the following one-way key exchange protocol useful for email applications: A computes $r_A = g^{k_A} \bmod p$ with random $k_A \in \mathbf{Z}_q$ and the session key $K = h(y_B^{k_A} \bmod p, r_A, d)$, encrypts a message m as $c = E_K(m)$ and sends $\{r_A, d, c\}$ to B, where d is a timestamp. B can then compute $K = h(r_A^{x_B} \bmod p, r_A, d)$ and decrypt c. In this protocol A may send $r_A = -g^{k_A} \bmod p$. If B does not respond or claims a garbage mail, then A knows that x_B is odd. This attack may be repeated t times, revealing the last t bits of x_B, if $2^t | p-1$.

3 Extracting Secret Keys in Other DL-based Schemes

There are many other discrete logarithm-based protocols which may be susceptible to our attack. In this section we present several such examples that we have found in the literature. They include threshold cryptosystems based on ElGamal encryption [15], anonymous channels used in electronic voting schemes [29, 35] and undeniable signatures [12, 8, 28].

3.1 Shared Decryption of ElGamal Encryption

ElGamal encryption of message m for user A consists of $\{c_1, c_2\}$, where $c_1 = g^k \bmod p$ with $k \in_R \mathbf{Z}_q$ and $c_2 = m y_A^k \bmod p$ [18]. The receiver A can decrypt the ciphertext $\{c_1, c_2\}$ by computing $m = c_2 c_1^{-x_A} \bmod p$. In some group-oriented applications we may need to encrypt the message in such a way that only an authorized subset of receivers can decrypt the ciphertext. This can be done using ElGamal encryption and Shamir's secret sharing scheme [36].

As an example, we consider a prime field implementation of the threshold cryptosystem proposed by Desmedt and Frankel [15]. Let $\mathbf{G}$ be a group of n members and $y_G = g^{x_G} \bmod p$ be a public key of the group. We want to encrypt a message m so that any subset of t or more members in $\mathbf{G}$ can read the message. For this, in the system setup phase a trusted authority picks a random polynomial f of degree $t-1$ in $\mathbf{Z}_q$ such that $f(0) = x_G$, i.e., $f(z) = a_{t-1}z^{t-1} + \cdots + a_1 z + x_G$ with $a_j \in_R \mathbf{Z}_q$, computes secret shares $x_{Gi} = f(i) \bmod q$ for $i = 1, 2, \cdots, n$ and securely sends x_{Gi} to each member i of $\mathbf{G}$. (See [24] for a more flexible scheme not requiring such pre-distribution of secret shares.) Now, suppose that a ciphertext $\{c_1, c_2\}$,where $c_1 = g^k \bmod p$ and $c_2 = my_G^k \bmod p$, is received and that a subset $\mathbf{H}$ of t members in $\mathbf{G}$ agreed to decrypt the ciphertext. Then each member $j \in \mathbf{H}$ computes $w_j = c_1^{-b_j x_{Gj}} \bmod p$, where $b_j = \prod_{i \in \mathbf{H}, i \neq j} \frac{-i}{j-i} \bmod q$, and sends w_j to a combiner (e.g., one designated member). The combiner then computes $w = \prod_{j \in \mathbf{H}} w_j \bmod p$, which should be $c_1^{-x_G} \bmod p$ if all members involved worked correctly. Therefore, the message m can be recovered by $m = c_2 w \bmod p$.

It is easy to see that our attack can be successful for the above scheme, if each shareholder does not check that $c_1^q = 1 \bmod p$. In this case, our attack can extract much more secret bits at a time. Let $\gamma = \prod \beta_i$ (a product of smooth order elements). The attacker sends a ciphertext $\{c_1, c_2\}$ such that $c_1 = \gamma g^k \bmod p$ and $c_2 = my_G^k \bmod p$. Since $w_j^q = (\gamma^q)^{-b_j x_{Gj} \bmod q} \bmod p$, once obtaining w_j, he can easily compute the logarithm $(-b_j x_{Gj} \bmod q) \bmod ord(\gamma)$ using a Pohlig-Hellman decomposition. The remaining part of $-b_j x_{Gj} \bmod q$ can be found from the value $y_j = g^{x_{Gj}} = w_j^{b_j k} \bmod p$. This reveals the secret share of a shareholder j. If w_j's are transmitted through a secure channel, the attacker need to collude with the combiner. (The combiner needs the help of the attacker to compute y_j's if they are not publicly available.)

Note the efficiency of the above attack. Unlike in key exchange protocols, where the attacker can only obtain a function of the shared secret (e.g., a hash value), in the above scheme the attacker has direct access to the shared secret itself (i.e., a value exponentiated with the secret key). This allows the attacker to get a Pohlig-Hellman decomposition for the secret key. Since now Pollard's ρ-method can be used to solve the decomposed problems, it would be quite feasible to use a β of order about 80 bits. Thus, for a random prime p such that $q|p-1$ and $|q| = 160$, the attack could reveal the whole secret key in most cases.

Anonymous channels proposed by Park et al.[29] uses a special case of the threshold cryptosystem described above, i.e., the case of $t = n$. The anonymous channel is primarily used to protect the secrecy of votes in electronic voting schemes. Later Pfitzmann [31] developed successful attacks on these channels. To defeat such attacks, Sako and Kilian [35] used a prime order subgroup in their election scheme, instead of the full multiplicative group $\mathbf{Z}_p^*$ originally used in [29]. However, in this case our attack can be applied again. To see this, we briefly describe the modified version in [35].

Each MIX M_i ($1 \leq i \leq n$) has a secret key $x_i \in_R \mathbf{Z}_q$ and publishes its public key $y_i = g^{x_i} \bmod p$. Let $w_j = \prod_{i=j+1}^{n} y_i \bmod p$ for $j < n$ and $w_n = 1$. For each ciphertext $C_0 = \{c_{0,1}, c_{0,2}\} = \{g^k, mw_0^k\}$, each MIX M_i for $i = 1, 2, \cdots, n-1$ transforms the C_{i-1} posted by M_{i-1} into $C_i = \{c_{i-1,1}g^{r_i}, c_{i-1,2}w_i^{r_i}c_{i-1,1}^{-x_i}\}$ with random $r_i \in \mathbf{Z}_q$ and posts the C_i on the public board in alphabetical order (all computations are done in mod p). In [35] this is done in two phases: in the first phase M_i posts $z_i = c_{i-1,1}^{x_i} \bmod p$ and in the second phase it posts $\{c_{i-1,1}g^{r_i}, c_{i-1,2}w_i^{r_i}/z_i\}$. In each phase M_i also proves the correctness of the computation. Now the final MIX M_n can recover m by computing $c_{n-1,2}c_{n-1,1}^{-x_n} \bmod p$.

As is clear, this protocol is vulnerable to our attack. A voter V can submit a faked vote $C_0 = \{\gamma c_{0,1} \bmod p, c_{0,2}\}$ and then find $x_i \bmod ord(\gamma)$ from z_i as before. Therefore, it is essential that each MIX M_i should verify that $c_{i-1,1}^q = 1 \bmod p$ before beginning its processing.

3.2 Undeniable Signatures

Our attack can also be applied to some digital signature applications. The most obvious case is to produce an undeniable signature on message $m \in \mathbf{Z}_q$ as $m^x \bmod p$ without checking that $m^q = 1 \bmod p$, where x is the signer's secret key. We could find several examples in the literature.

As a first example, we consider the validator issuing protocol by Chaum and Pedersen (see Sec.4 in [13]). The purpose of this protocol is that a center Z issues a validator to a 'wallet with observer' (consisting of a computer C and a tamper-proof module T embedded inside C). The validator is an unlinkable certificate for the public key $y_T = g^{x_T}$ of T. In some steps of the protocol the computer C blinds y_T as $m = y_T^k \bmod p$ with random k and sends it to the center Z, who then returns $z_0 = m^{x_Z} \bmod p$. Obviously, if C sends $m = \gamma y_T^k \bmod p$ with $\gamma = \prod \beta_i$, it can find $x_Z \bmod ord(\gamma)$ from $z_0^q = (\gamma^q)^{x_Z} \bmod p$ using the Pohlig-Hellman method. Note that C can still obtain the desired signature by computing $z_0\gamma^{-x_Z} \bmod p$ after finding $x_Z \bmod ord(\gamma)$. The same attack can be applied to its privacy enhanced version [14] if the signer does not check that $m^q = 1 \bmod p$. The authors may omit this checking step in the thought that C can only obtain an undeniable signature for a random message, but this omission causes a fatal attack as above.

In Brands's electronic cash scheme using a wallet with observers [6] (see [7] for more details), each user computes $I = g_1^u \bmod p$ with $u \in_R \mathbf{Z}_q$ and sends it to the bank, which generates a signature $z = (Ig_2)^x \bmod p$ (g_1, g_2 generators of a subgroup of order q). In this case the user must prove to the bank that he knows u since I corresponds to the account number of the user (see also [11]). Thus our attack is not applicable here. However, as noted in Sec.2.1, it is essential to check $I^q = 1 \bmod p$ at the begining of the proof if a Schnorr-type identification scheme is used for this purpose (this is the case in [7]). Otherwise, the user can pass the proof with $I = \beta g_1^u \bmod p$ in success probability $1/ord(\beta)$. The successful pass will be fatal in this system: Not only the user can mount our attack to find partial information on the secret x, but also he can spend the same coin multiple times without being identified.

Another possibility for the attack exists in the confirmation protocol of undeniable signatures [12, 8] and designated confirmer signatures [28]. For example, consider the convertible undeniable signature scheme by Boyar et al.[8]. In this scheme the signer S possesses two secret/public key pairs, $\{x \in_R \mathbf{Z}_q, y = g^x \bmod p\}$ and $\{z \in_R \mathbf{Z}_q, u = g^z \bmod p\}$. The signature on message m is a triple $\{t, r, s\}$, where $t = g^{k_1} \bmod p$, $r = g^{k_2} \bmod p$ and $s = k_2^{-1}(h(m)tzk_1 - xr) \bmod q$ ($k_1, k_2 \in_R \mathbf{Z}_q$). Thus the signature $\{t, r, s\}$ is valid iff $(t^{h(m)t})^z = y^r r^s \bmod p$. The confirmation protocol between S and V is as follows:

1. S and V computes $w = t^{h(m)t} \bmod p$ and $v = y^r r^s \bmod p$ from the signature $\{t, r, s\}$.
2. V computes a challenge $ch = w^a g^b \bmod p$ with $a, b \in_R \mathbf{Z}_q$ and sends ch to S.
3. S computes $h_1 = ch \cdot g^c \bmod p$ with $c \in_R \mathbf{Z}_q$ and $h_2 = h_1^z \bmod p$, and sends $\{h_1, h_2\}$ to V.
4. V reveals a and b to S.
5. S checks that $ch = w^a g^b \bmod p$. if it holds true, then S reveals c to V. Otherwise, S stops the protocol.
6. V checks that $h_1 = w^a g^{b+c} \bmod p$ and $h_2 = v^a u^{b+c} \bmod p$.

This protocol is complete, sound and is known to be zero-knowledge. However, suppose that the verifier V sends, as a challenge in step 2, any value of order q multiplied by small order elements, say $ch = \gamma g^b \bmod p$. Then the received value h_2 in step 3 satisfies $h_2^q = (\gamma^q)^z \bmod p$, from which he can find $z \bmod ord(\gamma)$. This shows that the confirmation protocol cannot be zero-knowledge against a dishonest verifier. It is essential for S to check that $ch^q = 1 \bmod p$ in step 3. In a variant by Pedersen [30], S computes h_1, h_2 as $h_1 = (ch)^c \bmod p$ with $c \in_R \mathbf{Z}_q$ and $h_2 = h_1^z \bmod p$. This variant is also vulnerable to our attack, since one can still obtain the equation $h_2^q = (h_1^q)^z \bmod p$ by sending $ch = \gamma g^b \bmod p$ (here note that $ord(h_1^q) = ord(\gamma)$).

The above attack suggests that a prime p should be chosen as $p = 2q + 1$ (q prime) if an undeniable signature is computed as $m^x \bmod p$ as in Chaum's undeniable signature [12], since there is no way to detect our attack otherwise.[5] (Note that it is infeasible to generate m as an element of order q for any meaningful message or its hash value if q is chosen small compared to p.) However, Jakobsson and Yung [19] failed to observe this fact when choosing system parameters in Chaum's scheme: $p = ql + 1$ (p, q prime, l integer), g a generator of G_q[6] and $\{x \in_R \mathbf{Z}_q, y = g^x \bmod p\}$ as the secret/public key pair of the signer. Careful examination shows that their oblivious protocol for deciding an undeniable signature is also vulnerable to our attack. This protocol cannot be repaired by simply adding checking steps as noted above. To avoid our attack, we have to choose p as $p = 2q + 1$ (q prime). Or we may use the full range of exponents.

We note that the signer or the confirmer(s) must verify the validity of a signature before executing the confirmation protocol. Otherwise, the verifier may change the first component t of the signature as $t' = \gamma t \bmod p$ and requests S to confirm its validity by sending $\{t', r, s\}$. Then V will be able to obtain $\gamma^z \bmod p$ at the end of the protocol, since the check in step 5 will succeed if V sends $ch = (w')^a g^b \bmod p$ with $w' = (t')^{t' h(m)} \bmod p$ in step 2. (see also Appendix A in [19] for another weakness when the validity of signatures is not checked before confirmation or denial.) If the secret z is distributed into a set of designated confirmers [30, 28], an authorized subset of confirmers should run a shared verification protocol to validate the signature. Then there may exist another possibility for the combiner to mount a similar attack to find secret shares of confirmers as in the shared decryption of ElGamal ciphertexts (see Sec.3.1), unless each confirmer checks that $w^q = 1 \bmod p$.

4 Generating and Registering Public Parameters

The presented attack shows that it is essential in discrete log-based schemes using a prime order subgroup to verify that received numbers belong to the underlying group before applying the secret key. However, this checking step requires one exponentiation, which seems too much in most applications. The best alternative is to use a prime p such that $(p-1)/2q$ is also prime or each prime factor of $(p-1)/2q$ is larger than q. (see also [25] for a method of generating primes which can substantially reduce the modular reduction time and storage usage). Such a prime can be generated much faster than a safe prime (i.e., a prime of the form $p = 2q + 1$).

To generate a prime p such that $p = 2qp_1 + 1$, we first choose a random prime p_1 of length $|p| - |q| - 1$ and then find a desired prime p by testing $p = 2qp_1 + 1$ for primality with random primes q's. Thus we need to generate a number of q's to find a p (e.g., about 710 for a 1024-bit

[5] Of course, our attack is useless if a base element is taken as a primitive element mod p. The public key $y = \alpha^x \bmod p$ already allows a Pohlig-Hellman decomposition, revealing smooth part of x.

[6] In [19] it is stated that g is a generator of G_p. We assume that G_p is a misprint for G_q. The choice of g as a primitive element mod p and the restriction of exponents to $\mathbf{Z}_q$ will be already insecure against the Pohlig-Hellman attack. See [40] for details.

prime p, considering the density of primes, $1/\ln x$). However, this does not require much time, since the size of q is usually small (e.g., 160 bits).

It is much cheaper to generate a prime p such that $p = 2qp_1p_2\cdots p_n + 1$, where p_i's are primes almost equal to q. We first determine the number n from the inequality $l = |p_i| \approx (|p|-|q|-1)/n \geq |q|$. Then we generate a pool of primes for p_i's. Suppose that the pool contains m primes of size l. Then we have $\binom{m}{n}$ candidates for p. Considering the density of primes, we can make this number large enough to guarantee that there are enough possibilities for a prime p to be found with this prime pool. For example, for a 1024-bit prime p and a 160-bit prime q we may choose $l \approx 173$, $n = 5$ and $m = 15$. This choice of parameters gives about 3000 candidates for p. Thus testing this many candidates will give a prime p with very high probability.

The latter form of primes will be sufficient to defeat our attack and can be generated much faster than the former form of primes (though somewhat slower than a random prime p with $q|p-1$). Even when we work in the full multiplicative group $\mathbf{Z}_p^*$, such a prime seems as strong against any known attack as a safe prime. Note that if protocol variables are not properly checked, an attacker can always acquire at least one bit of the secret, the parity bit, by our attack. We may trade off this one bit loss with efficiency by using a secure prime.

There are other advantages in our proposed prime generating methods. They give a complete factorization of $p-1$, which may be useful if we need to find a primitive element mod p. Note that there is no known algorithm to find a primitive element mod p without knowing the factorization of $p-1$. The simplest is to check the order of elements successively (say, $2, 3, 5, \cdots$) using the prime factors of $p-1$. In the latter form of primes, we can make $p-1$ contain many prime factors of similar size, and thus we may use different subgroups of prime order for different applications (e.g., one for ordinary signatures, one for undeniable signatures and one for key exchange, etc.). This will prevent any potential weakness from the misuse of key parameters. Of course, using the same prime p in different primitives may not be desirable in view of security against most discrete logarithm algorithms such as the index calulus method and the number field sieve. However, this gives us efficiency in storage and communication.

The attacks presented in this paper and in Menezes et al.[27] show that the CA must check for knowledge of a secret key corresponding to the public key before issuing a certificate. In particular, the CA and a user must first check the order of the received base g and public key y. If such an interactive proof is hard to carry out in some environments, we propose to use the following certification procedure, which seems to preclude any known weakness even without such a proof. The basic idea is to allow the CA to also contribute to the randomness of a user's secret key. On receiving a user's part of a public key $y' = g^{x'} \bmod p$, the CA computes the actual public key y as $y = (y')^a g^b \bmod p$, where $a = \frac{p-1}{q}c$ and $b, c \in_R \mathbf{Z}_q$. The CA now generates a certificate for y and sends $\{a \bmod q, b\}$, along with the certificate, to the user. The user can then compute the actual secret key x for the certified public key y as $x = x'a + b \bmod q$. The exponent a makes vanish any component of y' which does not belong to the subgroup of order q. This prevents users from registering an improper form of public keys. The multiplicative factor $g^b \bmod p$ makes it impossible for a user to register a public key as a power of other user's public key. This prevents the attack presented in [27]. Furthermore, users' secret keys can be made more pseudorandom by the CA's contribution if the CA uses a cryptographically strong pseudorandom generator (in this case we assume that the CA sends $\{a \bmod q, b\}$ through a secure channel). This may be advantageous since most users may not be so careful in choosing their secrets.

5 Conclusion

We have presented a key recovery attack on discrete logarithm-based protocols working in a prime order subgroup. This attack enables us to find part of, in many cases the whole, secret key of a victim in a reasonable time for many DL-based schemes published in the literature which disclose a shared secret or any function of it during the protocol execution. The attack uses small order subgroups in $\mathbf{Z}_p^*$ to compute part of the secret key in a protocol working in a subgroup of prime order q. This is possible since in most schemes a prime p is chosen at random such that $q|p-1$. Therefore, our attack can be easily prevented if relevant protocol variables are properly checked, that is, if each party checks that received numbers belong to the underlying subgroup of prime order. However, such checkings substantially decrease efficiency. Thus a better alternative would be to minimize possible leakage of secret key bits by using a secure prime, a prime p such that all prime factors of $(p-1)/2q$ are larger than q. Such a prime only leaks one bit of the secret by our attack.

References

[1] R.Anderson and R.Needham, Robustness principles for public key protocols, In *Advances in Cryptology - CRYPTO'95*, LNCS 963, Springer-Verlag, 1995, pp.236-247.

[2] R.Anderson and S.Vaudenay, Minding your p's and q's, In *Advances in Cryptology - ASIACRYPT'96*, LNCS 1163, Springer-Verlag, 1996, pp.15-25.

[3] A.Aziz, T.Markson and H.Prafullchandra, Simple key-management for Internet protocols (SKIP), *draft-ietf-ipsec-skip-07.txt*, Aug. 1996. (see also the SKIP home page *http : //skip.incog.com/* for more information.)

[4] D.Bleichenbacher, Generating ElGamal signatures without knowing the secret, In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, Springer-Verlag, 1996, pp.10-18.

[5] C.Boyd and W.Mao, Design and analysis of key exchange protocols via secure channel identification, In *Advances in Cryptology - ASIACRYPT'94*, LNCS 917, Springer-Verlag, 1995, pp.171-181.

[6] S.Brands, Untraceable off-line cash in wallet with observers, In *Advances in Cryptology - CRYPTO'93*, LNCS 773, Springer-Verlag, 1994, pp.302-318.

[7] S.Brands, An efficient off-line electronic cash system based on the representation problem, *Technical Report CS-R9323*, CWI, Amsterdam, 1993.

[8] J.Boyar, D.Chaum, I.Damgard and T.Pedersen, Convertible undeniable signatures, In *Advances in Cryptology - CRYPTO'90*, LNCS 537, Springer-Verlag, 1991, pp.189-205.

[9] M.Burmester, A remark on the efficiency of identification schemes, In *Advances in Cryptology - EUROCRYPT'90*, LNCS 473, Springer-Verlag, 1991, pp.493-495.

[10] M.Burmester, On the risk of opening distributed keys, In *Advances in Cryptology - CRYPTO'94*, LNCS 839, Springer-Verlag, 1994, pp.308-317.

[11] A.Chan, Y.Frankel and Y.Tsiounis, Mis-representation of identities in E-cash schemes and how to prevent it, In *Advances in Cryptology - ASIACRYPT'96*, LNCS 1163, Springer-Verlag, 1996, pp.276-285.

[12] D.Chaum, Zero-knowledge undeniable signatures, In *Advances in Cryptology - EUROCRYPT'90*, LNCS 473, Springer-Verlag, 1991, pp.458-464.

[13] D.Chaum and T.Pedersen, Wallet databases with observers, In *Advances in Cryptology - CRYPTO'92*, LNCS 740, Springer-Verlag, 1993, pp.89-105.

[14] R.Cramer and T.Pedersen, Improved privacy in wallets with observers, In *Advances in Cryptology - EUROCRYPT'93*, LNCS 765, Springer-Verlag, 1994, pp.329-343.

[15] Y.Desmedt and Y.Frankel, Threshold cryptosystems, In *Advances in Cryptology - CRYPTO'89*, LNCS 435, Springer-Verlag, 1990, pp.307-315.

[16] W.Diffie and M.E.Hellman, New directions in cryptography, *IEEE Trans. Info. Theory*, 22(6), 1976, pp.644-654.

[17] W.Diffie, P.van Oorschot and M.Wiener, Authentication and authenticated key exchange, *Designs, Codes and Cryptography*, 2, 1992, pp.107-125.

[18] T.ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inform. Theory*, IT-31, 1985, pp.469-472.

[19] M.Jakobsson and M.Yung, Proving without knowing: on oblivious, agnostic and blindfolded provers, In *Advances in Cryptology - CRYPTO'96*, LNCS 1109, Springer-Verlag, 1996, pp.186-200.

[20] M.Just and S.Vaudenay, Authenticated multi-party key agreement, In *Advances in Cryptology - ASIACRYPT'96*, LNCS 1163, Springer-Verlag, 1996, pp.36-49.

[21] H.Krawczyk, SKEME: A versatile secure key exchange mechanisms for Internet, In *Proc. of 1996 Symp. on Network and Distributed Systems Security.*

[22] A.K.Lenstra, P.Winkler and Y.Yacobi, A key escrow system with warrant bounds, In *Advances in Cryptology - CRYPTO'95*, LNCS 963, Springer-Verlag, 1995, pp.197-207.

[23] C.H.Lim and P.J.Lee, Several practical protocols for authentication and key exchange, *Information Processing Letters*, 53, 1995, pp.91-96.

[24] C.H.Lim and P.J.Lee, Directed signatures and application to threshold cryptosystems, In *Pre-Proc. of 1996 Cambridge Workshop on Security Protocols*, The Isaac Newton Institute, Cambridge, April 1996.

[25] C.H.Lim and P.J.Lee, Generating efficient primes for discrete log cryptosystems, submitted for publication (also presented at ASIACRYPT'96 Rump Session).

[26] T.Matsumoto, Y.Takashima and H.Imai, On seeking smart public-key distribution systems, *The Transactions of the IEICE of Japan*, E69, 1986, pp.99-106.

[27] A.J.Menezes, M.Qu and S.A.Vanstone, Some new key agreement protocols providing implicit authentication, In *Proc. SAC'95*, Carleton Univ., Ottawa, Ontario, May 1995, pp.22-32.

[28] T.Okamoto, Designated confirmer signatures and public-key encryption are equivalent, In *Advances in Cryptology - CRYPTO'94*, LNCS 839, Springer-Verlag, 1995, pp.61-74.

[29] C.S.Park, K.Itoh and K.Kurosawa, Efficient anonymous channel and all/nothing election scheme, In *Advances in Cryptology - EUROCRYPT'93*, LNCS 765, Springer-Verlag, 1994, pp.248-259.

[30] T.Pedersen, Distributed provers with applications to undeniable signatures, In *Advances in Cryptology - EUROCRYPT'91*, LNCS 547, Springer-Verlag, 1991, pp.221-242.

[31] B.Pfitzmann, Breaking an efficient anonymous channel, In *Advances in Cryptology - EUROCRYPT'94*, LNCS 950, Springer-Verlag, 1995, pp.332-340.

[32] S.C.Pohlig and M.E.Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Inform. Theory*, IT-24 (1), 1978, pp.106-110.

[33] D.Pointcheval and J.Stern, Security proofs for signature schemes, In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, Springer-Verlag, 1996, pp.387-398.

[34] J.M.Pollard, Monte Carlo methods for index computation (mod p), *Math. Comp.*, 32(143), 1978, pp.918-924.

[35] K.Sako and J.Kilian, Receipt-free mix-type voting scheme, In *Advances in Cryptology - EUROCRYPT'95*, LNCS 921, Springer-Verlag, 1995, pp.pp.393-403.

[36] A.Shamir, How to share a secret, *Commun. ACM*, 22, 1979, pp.612-613.

[37] C.P.Schnorr, Efficient identification and signatures for smart cards, In *Advances in Cryptology - CRYPTO'89*, LNCS 435, Springer-Verlag, 1990, pp.235-251.

[38] J.Stern, The validation of cryptographic algorithms, In *Advances in Cryptology - ASIACRYPT'96*, LNCS 1163, Springer-Verlag, 1996, pp.301-310.

[39] P.C.van Oorschot and M.J.Wiener, Parallel collision search with applications to hash functions and discrete logarithms, In *Proc. 2nd ACM Conference on Computer and Communications Security*, Fairfax, Virginia, Nov. 1994, pp.210-218.

[40] P.C.van Oorschot and M.J.Wiener, On Diffie-Hellman key agreement with short exponents, In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, Springer-Verlag, 1996, pp.332-343.

[41] S.Vaudenay, Hidden collisions on DSS, In *Advances in Cryptology - CRYPTO'96*, LNCS 1109, Springer-Verlag, 1996, pp.83-88.

[42] Y.Yacobi, A key distribution paradox, In *Advances in Cryptology - CRYPTO'90*, LNCS 537, Springer-Verlag, 1991, pp.268-273.

[43] ISO/IEC JTC1/SC27, Information technology - Security techniques - Key management - Part 3: Mechanisms using asymmetric techniques.