Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 5 (9): 1692-1712, 2005
ISSN 1812-5654
© 2005 Asian Network for Scientific Information

Security of the Cryptographic Protocols Based on Discrete Logarithm Problem

Kefa Rabah
Department of Physics, Eastern Mediterranean University,
Gazimagusa, North Cyprus, via Mersin 10, Turkey

Abstract: We wish to find the smallest non-negative integer, β, for which $y=g^{\beta}$ where, $y, \beta \in GF(p)$ (if such an β exists). This is the Discrete Logarithm Problem (DLP). A number of strategies have been proposed to solve the DLP, among them, Shanks Baby-Step Giant-Step algorithm, the Pollard Rho algorithm, the Pohlig-Hellman algorithm and the Index-Calculus method. We show that, given certain assumptions about the smoothness of the integers, the index calculus will, in general, out-perform the other three methods, substantially increasing the range of problems which are feasible to solve and thereby threatening the security of the DLP-based crypto-algorithms like, DH key exchange protocol, ElGamal cryptosystem, DSA and many others. In this paper we describe basic principle and implementation procedure to these DLP-crypto algorithms. We will also discuss the general methods of attacking DLP cryptosystems and how secure they are against these general attacks. The mathematical challenge here lies in computing discrete logarithms in finite fields of type Z_p, which consist of the integers modulo a large prime p.

Key words: Public-key cryptography, DLP, Diffie-Hellman, digital signature, ElGamal, internet security, attacks, message digest, authentication, index calculus, factoring

INTRODUCTION

Discrete logs have a long history in number theory. Initially, they were used primarily in computations in finite fields (where they typically appeared in the closely related form of Zech's logarithm). However, they were rather obscure, just like Integer Factorization Problem (IFP). Unlike the latter, they could not even invoke any famous quotes of Gauss[1] about their fundamental importance in mathematics. The status of discrete logs started to grow in the 20th century, as more computations were done and as more thought went into algorithmic questions. It appears that they started to play an important role in cryptography already in the 1950s, long before public key systems appeared on the scene, as cryptosystems based on shift-register sequences displaced those on rotor machines[2]. Discrete logs occur naturally in the context as tools for finding where in a shift register sequence a particular block occurs. The main impetus for the intensive current interest in discrete logs, though, came from the invention of the Diffie-Hellman method[3].

The most important tool necessary for the implementation of public-key cryptosystems is the Discrete Log Problem (DLP). Many popular modern crypto-algorithms base their security on the DLP[1,2]. Based on the difficulty of this problem, Diffie-Hellman[3,4] proposed the well-known Diffie-Hellman key agreement scheme in 1976. Since then, numerous other cryptographic protocols whose security depends on the DLP have been proposed, including: the ElGamal encryption and signature scheme[5], the US government Digital Signature Algorithm (DSA)[6-9] is perhaps the best known example of a DLP system, the Schnorr signature scheme[10] and the Nyberg-Reuppel signature scheme[11,12]. Due to interest in these applications, the DLP has been extensively studied by mathematicians for the past 20 years. The mathematical challenge here lies in computing discrete logarithms in finite fields of type Z_p, which consist of the integers modulo a large prime p. Although this problem can be considered difficult, there are known sub-exponential time algorithms for solving it, such as the, index calculus[12] and Number Field Sieve (NFS)[13]. In practical terms, sub-exponential time means that a determined hacker with enough processing power can break the system in a few months.

THE MECHANICS OF DISCRETE LOG PROBLEM

In an (abelian) group G^* (multiplicatively written) we can consider the equation $y=x^n$, $x, y \in G$, $n \in Z$. If x and y are known real numbers and it is also known that x is some power (say, n) of y, then logarithms can be used to find n("$=\log_x y$") in an efficient manner (here n is known as the index of $y \in G$). However, if x and y are given such that: $y= x^n = x \cdot x \dots x$ (n-times), then in general it is technically much harder and hence the determination of n cannot be carried out in a reasonable amount of time. This is equivalent to the well-known real logarithm; we call n the discrete logarithm of y related to the base x[10]. The operation exponentiation $x \rightarrow y := x^n$ can be implemented as a quick, efficient algorithm. As an example, the exponentiation computation of 5^e, can be performed pretty efficiently by using binary expansion of the exponent e, for example, for $e = 4369_{10} = 1000100010001_2$, we have:

$$5^{4369} = \left(\left(\left(\left(\left(5^{2^4}\right)5\right)^{2^4}\right)5\right)^{2^4}\right)5$$

The DLP can also be implemented in modular arithmetic form and can be defined as follows: given a prime p, a generator g of Z_p and a non-zero element $\gamma \in Z_p$, find the unique integer k, $0 \le k \le p-2$, such that $\gamma = g^k \bmod p$. The integer k is called the discrete logarithm of γ to the base g (which we can write smartly as follows: $b^e = r(\bmod p)$; $b^e = r$ and finally $e = \log_b r$, where, b is the base, e is the exponent and r is the residual mod p). Here, Z_p denotes the set of integers $\{0, 1, 2, \cdots, p-1\}$, where, addition and multiplication are performed modulo p. It is well-known that there exists a non-zero element $g \in Z_p$ such that each non-zero elements in Z_p can be written as a power of g; such an element g is called a generator of Z_p.

Similarly, we can perform a modular exponentiation easily, for example, the computation of $5^e \bmod p$, can be carried out efficiently: after each squaring or after each multiplication by 5 reduced modulo p and then continue. That is, if p = 5779, then, $5^e \bmod p = 4720$.

On a similar note, we can easily solve, $5^e \equiv 2437 \pmod{5779}$, which is equivalent to determining: $e = \log_5 2437$ in Z_{5779} and there is no known method with a similar low complexity. That is, it is easy to take logarithm but not a modular/discrete logarithm. This is the backbone of the crypto-algorithm based discrete logarithm problem[1].

As an example, let's solve: $b^{4369} \equiv 2437(\bmod 5779)$. To solve our DLP, we note that p = 5779 is indeed a prime number, so that by Fermat little theorem: $a^{p-1} \equiv 1(\bmod p)$ for $a \in (Z/nZ)^*$, we have: $b^{5778} \equiv 1(\bmod 5779)$. It follows that, if we raise both sides of our equation to the power of 529 (i.e., the multiplicative inverse: $e^{-1}(\bmod p\text{-}1) = (4369)^{-1} \bmod 5778 = 529$), we find the solution of our DLP: $b \equiv r^{1/e}(\bmod p) \equiv 2437^{529}$ (mod 5779). = 2249.

In general, if the modulus of DLP is replaced with a product of two primes, then finding the solution becomes naturally infeasible for large moduli, simply because the factorization of large integer number is infeasible. This is the backbone on which the security of public key like RSA cryptosystems[7,14]. In RSA public key cryptosystem, for example, Bob's public key is (e, n) and his private key is (d,n) where, n is the product of two prime numbers p and q (i.e., $n(= p \cdot q)$) such that: $ed = 1(\bmod(p\text{-}1)(q\text{-}1))$. The product, n, is the modulus, e is the public exponent and d is the secret exponent. To encrypt a plaintext message M for Bob, Alice has to compute ciphertext: $C = M^e(\bmod n)$. Bob can decrypt C by computing: $(C)^d = (M^e)^d = M(\bmod n) = M$. No one except Bob can decrypt C since d is only known to Bob. The RSA crypto-algorithm can be broken by factoring n into p and q. If n is factored then (p-1)(q-1) can be found and from this d can be computed. Hence, any adversary that factors n can find the private-key $d \equiv e^{-1} (\bmod (p\text{-}1)(q\text{-}1))$ and with it decrypt any encrypted message[7,14]. Therefore, the algorithm is secure only if the factorization of the carefully chosen sufficiently large two prime numbers requires a super-polynomial amount of time with respect to the size of the number. The key question is, therefore, how large is sufficiently large to make this recovery virtually impossible? In the 1980s it was generally held that prime numbers of a fifty odd digits (i.e., 10^{50}) would suffice. Currently, you need a 1024-bit number to get the same security you got from a 512-bit number in the early 1980s[13]. If you want your keys to remain secure for 20 years, 1024 bits is probably too short (Fig. 1)[16].

Technical advantages between DLP-based crypto-algorithms and RSA is such that in many cases where algorithms of comparable functionality exist, say one over the finite field of integers modulo a prime p and another using composite integer n of the same size, breaking the discrete log modulo p appears to be somewhat harder than factoring the integer n. For purposes of key generation, many people prefer DH algorithm over RSA, since the session key it generates is evanescent. In the simplest application of RSA to key generation, as seen above, Alice creates a session key and transmits it to Bob using Bob's public key. An eavesdropper who can coerce Bob, via clever social engineering, into revealing his private key can recover the full text of the communication exchanged between Alice and Bob. On the other hand, if Alice and Bob use DH key exchange protocol to generate the session key, destroy it after the session ends and do not store their communication, then neither coercion nor cryptanalysis will enable the eavesdropper to recover what information was exchanged. It is widely believed that the DSA is based on the discrete logs because it is harder to use it for encryption than if it were based on RSA (and thus on Integer Factorization Problem, IFP).

Standard DLP cryptosystems are based on multiplicative groups with the main operation of exponentiation. The corresponding problem in additive

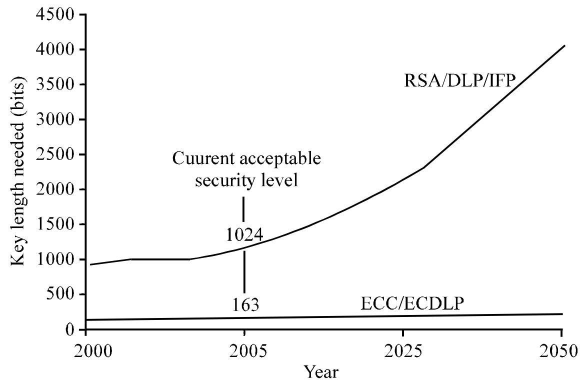

Fig. 1: Proposed the minimum key sizes (in bits) to be regarded as safe for RSA and ECC

(i.e., abelian) groups is: given P and kP = Q (P added to itself k times), find the integer k (i.e., find k = $\log_P Q$). This is much more difficult! There is no one-step operation like taking logarithms that we can use to get the solution. So we may know P and kP and yet not be able to find k in a reasonable amount of time. This is called the Discrete Log Problem for abelian groups. We could always repeatedly subtract P from kP till we get 0. But if k is large, this will take us a very long time! Several important cryptosystems are based on the difficulty of solving the DLP over finite abelian groups. The solution is even tougher if the underlying group arises from an elliptic curve over a finite field[16].

In 1985, Victor Miller (IBM[17]) and Neal Koblitz (University of Washington)[18], independently realized that the DLP-additive group associated with elliptic curve can be used for similar classical Diffie-Hellman (DH) public-key exchange method. In the same year Scott Vanstone and co-researchers, realized that the invention by Miller and Koblitz was not just a mathematical generalization of the original DH idea, but that some aspect of it could lead to a very promising alternative public key cryptosystems that could be put into real practical application-this lead to the formation of what is today, Certicom[19]. In Elliptic Curve Cryptography (ECC)[16], the multiplicative group is replaced by the additive group of elliptic curve points and exponentiation operation by scalar multiplication of a point (i.e. calculation of $g^k = g{\cdot}g....g$ (k-times) for a generator g of a multiplicative group is replaced by calculation of [k]P = P+P+.....+P (k-times) for a generator point P of an additive group of elliptic curve points). Thus, the computational performance of cryptographic protocols based on elliptic curves strongly depends on efficiency of the scalar multiplication. Further, Elliptic curve cryptosystems (ECC) appear to offer the possibility of using much smaller key sizes than would be required by RSA-type crypto-algorithms of comparable security (Fig. 1). For more detail on the implementation of ECC[16-19].

Time complexity of DLP: Suppose $G = Z^*_p$ with p a 200-bit prime. Using 2^{20} computers, each one running at 400 MHz and assuming that one exponentiation (b^e mod p) takes only one machine cycle! Then $400{\cdot}2^{20} \approx 2^{29}$ exponentiation can be made per second per computer. As there are around 2^{25} sec in a year, it would take about $2^{200}/(2{\cdot}2^{20+29+25}) \approx 2^{125}$ years to find the desired discrete logarithm, on average, by trial and error method.

The presumed intractability of the DLP, for appropriate choices of G, in contrast to the relative efficiency of calculating the discrete exponentiation, has made the DLP a basic building block of many cryptographic applications, including public-key encryption algorithms, digital signature schemes and key agreement protocols[2-6,11,16-19].

DIFFIE-HELLMAN MULTIUSER CRYPTOSYSTEM

Prior to 1970, symmetric key cryptosystems had been the crypto-mode in existence. In a symmetric key crypto-protocols, a common key (the master shared secrete key) are used by both communicating parties to encrypt and decrypt messages. These symmetric key crypto-protocols provide-high speed key and communication throughput but have the drawback that a common (or session) key must be established before communication between parties can be begin. The process of exchanging the crypto-keys is referred to as key distribution and can be very difficult[20]!

It was Merkle who first introduced the basic concept of public key cryptosystems with a view to overcome the key distribution problem[7,21]. However, it was W. Diffie and Hellman[3,4] who were the first to introduce practical public-key cryptography which eliminated the need for key distribution encountered with the private-key cryptosystems and it is widely used today. The system was discovered independently by GCHQ (British Intelligence) a few years before Diffie-Hellman found it, but couldn't tell anyone about their work; perhaps it was discovered by others before. That this system was discovered independently more than once shouldn't surprise you, given how simple it is! The encoding function here is a trapdoor function-one whose inverse is impractical to implement, unless some extra information is available. This extra information (called the decrypting-key) is not required for encrypting the message, yet is essential for decrypting it in reasonable time. The beauty of such a system is that the encrypting process need not be kept secret. Each user has his own or a personal encrypting-function, which is public information (hence the name public-key) and a decoding key, which he keeps secret.

The basic concept of public-key crypto-algorithm: In a public-key cryptosystems each user places in a public-key server an encryption procedure E. That is, the public-key server is a directory giving the encryption procedure of each user. The user keeps secret the details of his corresponding decryption procedure D. These procedures have the following properties:

a. Decrypting the encrypted form of plaintext message C = E(T) yields T, i.e.:

$$D(C) = D(E(T)) = T$$

b. Both E and D are easy to compute.
c. By publicly revealing E the user does not reveal an easy way to compute D. This means that in practice only he can decrypt messages encrypted with E, or compute D efficiently.
d. If a message T is first deciphered and enciphered, T is the result, i.e.:

$$E(D(T)) = T$$

An encryption (or decryption) procedure typically consist of a general method and an encryption key. The general method, under control of the key, encrypts a plaintext message T to obtain the form of the message or ciphertext *C*. Everyone can use the same general method; the security of a given message will rest on the security of the key. Revealing an encryption algorithm then means revealing the key.

When the user reveals E, he reveals a very inefficient method of computing D(C): testing all possible messages T until one finds E(T) = C. If property (c) is satisfied the number of such messages to test will be so large that this approach is impractical.

A function E satisfying (a)-(c) is a trap-door one-way function; if it also satisfies (d) it is a trap-door one-way permutation. In 1974 the first detailed description of such a one-way function was published[7,21]. That is, a one-to-one function f: $X \rightarrow Y$ is one-way if it is easy to compute a polynomial function f(x) for any $x \in X$ but hard to compute $f^{-1}(y)$ for most randomly chosen y in the range f. Diffie-Hellman[3,4] were the first to introduce the concept of trap-door one-way functions into crypto-algorithm. These functions are called one-way because they are easy to compute in one direction but (apparently) very difficult to compute in the other direction. They are called trap-doors functions since the inverse functions are in fact easy to compute once certain private trap-door information is known. A trap-door one-way function that also satisfies (d) must be a permutation: every message is the ciphertext for some other message and every ciphertext is itself a permissible message. (The mapping is one-to-one and onto). Property (d) is needed to implement digital signatures scheme[7,8]. Public key cryptosystems, however, tend to be more computation intensive than symmetric one, many of which are fully executed in hardware alone[20], but their algebraic foundations provide robust proofs of security that few symmetric crypto-schemes can match[2].

The mechanics of Diffie-Hellman algorithm: The Diffie-Hellman procedure depends on rather magical properties of whole numbers. In the nineteenth century Gauss established an elaborate body of theorems based on the idea of arithmetical remainders. He adopted a notation that is widely used by mathematicians today, y = x(mod n), what this means is that if you divide y by n you get a remainder x (cf. 15 = 2 mod 13). The symbol = (or ≡) is called a congruence relation and simply means equivalent to, while mod is short for modulus, or modulo, we will use both symbols interchangeably. If you multiply two numbers in this system you also still get a number between 1 and 13. For example, 4×6 mod 13 = 24 mod 13 = 11 mod 13. This then is just a notation.

Diffie-Hellman technique makes use of the apparent difficulty of computing logarithms over a finite field F_p with p number of elements. The systems parameters, therefore, consist of a large prime number p and a generator g of the multiplicative group Z^*_p whose powers modulo p generate a large number of elements. Let:

$$y = g^x (\text{mod } p) \qquad \text{for } 1 \le x \le p-1 \qquad (1)$$

where, g is a fixed primitive element of F_p, then x is arranged to as the logarithm of y to base g, modulo p:

$$x = \log_g y (\text{mod } p) \qquad \text{for } 1 \le y \le p\text{-}1 \qquad (2)$$

Calculation of y from x is easy, taking at most, $2 \cdot \log_2 p$, multiplications[22]. For example, x = 34:

$$y = g^{34} = ((((g^2)^2)^2)^2)^2 . g^{\ 2}$$

Computing x from y, on the other hand can be much more difficult and, for certain carefully chosen values of p, requires an $O(\sqrt{p})$, using the best known algorithm[23]. Security of DH, therefore, depends crucially on the security of computing logarithm modulo p and if an algorithm whose complexity grew as, $\log_2 p$, were to be found then DH crypto-security can be broken[1].

CLASSICAL DIFFIE-HELLAM KEY EXCHANGE PROTOCOL

Let us consider our usual communicating partners, Alice and Bob and don't forget the benevolent cracker Eve. Alice must communicate vital information to Bob that must reach him securely. Their communications are monitored by Eve, who must not discover the message. If Alice and Bob could agree on a secret encoding key, they could encrypt their message. Fortunately, Alice knows Diffie-Hellman algorithm.

Now let's apply this mathematical procedure to Diffie-Hellman algorithm. The method works as follows: (i) Both the active participants (say Bob and Alice) must first agree on two randomly generated prime numbers, p and q. Numbers p and q can be publicly known. Parameter p is a prime number and parameter g (usually called a generator) is an integer less than p, which is capable of generating every element from 1 to p-1 when multiplied by itself a certain number of times, modulo the prime p. (ii) Each participant must next choose another randomly generated number, perform a mathematical operation that involves p, q and the chosen number; and then transmit the result to the other participant.

Generation of shared key using DH key exchange protocol: The systems parameters consist of a large prime number p and a generator g of the multiplicative group Z^*_p whose powers modulo p generate a large number of elements. Alice and Bob agree on a prime number p and an integer g that has order p-1 modulo p.(So $g^{p-1} \equiv 1 \pmod p$)), but $g^{S_A} = 1 (\mathrm{mod}\, p)$ for any positive $s_A<p-1$.) Alice chooses a random number $s_A< p$ and Bob chooses a random number $s_B<p$. Alice sends $k_A = g^{S_A}$ (mod p) to Bob and Bob sends $k_B = g^{S_B}$ (mod p) Alice.

Alice (A) can now compute the secret-key:

$$k_{AB} \equiv (g^{S_A})^{S_B} \equiv g^{S_A S_B} (\mathrm{mod}\, p)$$

Likewise, Bob (B) computes the secret-key:

$$k_{BA} \equiv (g^{S_B})^{S_A} \equiv g^{S_B S_A} (\mathrm{mod}\, p)$$

One may notice that: $k=k_{AB} = k_{BA} = g^{S_A S_B} = g^s (\mathrm{mod} p)$, is the session or shared master secret key between Alice and Bob for future secure communication. The session-keys are the same (since $s = s_A s_B = s_B s_A$); hence Alice and Bob now have a shared master secret-key k. Future communications occur using the session key k. Thus, the session-key, k can be used with a private-key crypto-algorithms such as Data Encryption Standard (DES), 3DES or AES, for encryption purposes.

Now Alice uses the master secret key k to send Bob an encrypted version of her critical message. Bob, who also knows k, is able to decode the message. Meanwhile, hacker (Eve) see both, $g^{S_A} (\mathrm{mod}\, p)$ and $g^{S_B} (\mathrm{mod} p)$, but she aren't able to use this information to deduce either s_A, s_B or g^s (mod p) quickly enough to stop Bob from thwarting her plan. This is the backbone of the DLP crypto-security.

The only information that Eve knows is group G, g, g^{S_A} and g^{S_B}. If Eve can recover g^s from this data then Eve is said to solve Diffie-Hellman Problem (DHP). More specifically, we can define DHP as follows: Given a finite cyclic group G, with generator $g \in G$ and elements $\alpha, \beta \in G$. The DHP asks for $\gamma \in G$ such that: $\gamma = g^{((\log_g \alpha)(\log_g \beta))}$. It is easy to see that if Eve can find discrete logarithms in G (residue group modulo p) then she can solve DHP. It is believed for most groups in use in cryptography that DHP and the Discrete Log Problem (DLP) are equivalent in complexity-theoretic sense, there is a polynomial time reduction of one problem to the other and vice versa[2,24,25]. In any case, if one wishes to use the DHP in particular group as the basis of a cryptosystem, it is necessary that the DLP be hard in that group! Although there are many groups that have been proposed for which DHP may be hard and used securely-however, in practice there are only two that are most often used: One in multiplicative group $(F_q)^*$ of finite field of order q and which is employed in this study. Its slight modification is employed in Digital Signature Algorithm (DSA)[25].

However, it turns out that if the numbers are all sufficiently large, it is very hard to calculate the discrete logarithm in a reasonable time. The security of the Diffie-Hellman algorithm depends on this fact. Alternatively, the eavesdropper can have access to p, q, S_A and S_B but neither k_A nor k_B. As a result, the eavesdropper cannot calculate k.

If p is a prime slightly less than 2^n, then all quantities are representable as n-bit numbers. Exponentiation then takes at most 2n multiplications, mod p, while by hypothesis taking logs require, $q^{1/2} = 2^{1/2}$, operations. The cryptanalytic effort therefore grows exponentially relative to the computational efforts. If n = 200, then at most 400 multiplications are required to compute k_A from s_A, or k from k_A and s_B, yet taking logs, mod p, requires 2^{100} or approximately 10^{30} operations.

Implementation of DH key exchange protocol: In practice the systems parameters consist of a large prime number p and a generator g of the multiplicative group Z^*_p whose powers modulo p generate a large number of elements. For practical application and security reasons the crypto-keys must be of 1024-bit or greater is recommended (Fig. 1). However, here we consider an overly simple numbers to help us understand the basic implementation of DH procedure.

Systems parameters: The two communicating entities, Alice (A) and Bob (B), both selects the prime p = 12884901893(a 30-bit or 10 decimal digits) and a generator g = 12 of order p-1 = 12884901892; where, g, k_A, $k_B \in Z^*_p$.

Alice (A):

- Select a random integer $s_A = 2^{10}$ as her secret-key.
- Compute and sends to Bob:

$$k_A \equiv g^{S_A} \bmod p \equiv 12^{1024} (\bmod p) = 3505577916$$

Bob (B):

- Chooses a random integer $s_B = 2^{15} < p$ as his secret key
- Compute and sends to Alice:

$$k_B = g^{S_B} \bmod p = 12^{32768} (\bmod p) = 9663562615$$

Each entity compute shared master secret-key:

Alice: $k_{AB} = k = (k_A)^{S_B} \equiv 11093904324 (\bmod p)$
where shared session; $s = s_B s_A = s_A s_B = 33554432$;

Bob: $k_{BA} = k = (k_A)^{S_A} \equiv 11093904324 (\bmod p)$

Hence the session master secret key:
$k = k_{BA} = k_{AB} = g^s = 11093904324 \ (\bmod p)$

Indeed:
$$s = s_A s_B = \log_g y = \log_g 11093904324 = 33554432$$

The DH method is used for communication between two people and makes use of three keys: two secret-keys (one for each person) and a session key determined by the two people during the course of the conversation. In other words, the conversation starts with the two people using their own keys; they exchange information to determine a session key which is then used for all future messages. It is important to note that Diffie-Hellman algorithm is an excellent tool for key distribution, but cannot be used effectively to encrypt and decrypt messages on the fly independent of the person one communicates with (cf. email communication).

THE ELGAMAL ALGORITHM

The algorithm that we will use here is the ElGamal encryption algorithm. Taher ElGamal was the first mathematician to propose a public-key cryptosystem based on the Discrete Logarithm Problem (DLP)[4]. He in fact proposed two distinct cryptosystems: One for encryption and the other for digital signature scheme in 1984. Since then, many variations have been made on the digital signature system to offer improved efficiency over the original system. The ElGamal public-key encryption scheme can be viewed as Diffie-Hellman key agreement protocol in key transfer mode. Its security is based on the intractability of the discrete logarithm problem (DLP) and the Diffie-Hellman Problem (DHP)[3,4].

ElGamal encryption algorithm is very similar to the RSA encryption algorithm in the sense that it is a public key algorithm, which utilizes modular arithmetic on large numbers[7]. However, the mathematics involved is slightly more complicated. Let us start by examining what values constitute to both the public and private keys and how to generate them.

The systems parameters consist of a large prime number p and a generator g of the multiplicative group Z^*_p whose powers modulo p generate a large number of elements, as in Diffie-Hellman method. Let's assume the two entities Alice (A) and Bob (B) wants to communicate. Alice (A) generates a secrete (private-key) from a randomly chosen large integer number a such that $1 \leq a \leq p-2$ and computes her public-key A:

$$A = g^a (\bmod p) \qquad (3)$$

Alice's authentic public-key set is (p, g, A) and private-key (a, p). Note that g must be less than p and relative prime to p. This is essential to the algorithm because if a and p are not relatively prime, then we will have trouble using our mod operation.

Encryption: Bob (B) encrypts a message M for Alice (A), which A decrypts.

Suppose Bob wishes to send a plaintext message M to Alice, where, M is an integer in the range [0, p-1]:

- Bob first obtains A's authentic public-key ring: (p, g, A) placed in the public key server.
- Generates a large random integer b such that $1 \leq b \leq p-2$
- Computes his public-key: $B = g^b \pmod p$
- He uses DH key exchange protocol which they had agreed a prior to compute the shared masters secrete key: $S_{AB} = (A)^b \bmod p = g^{ab} \pmod p$
- Encrypts the plaintext message M: $\delta = M \cdot S_{AB} \pmod p$
- Sends the ciphertext $C = \{B, \delta\}$ to Alice.

Decryption: Upon receiving the ciphertext C, Alice uses her private-key a to compute:

$$S^{-1}_{AB} \equiv B^{p-1-a} \bmod p$$
$$(\text{not that}: S^{-1}_{AB} = B^{p-1-a} = B^{-a} = g^{-ab})$$

and recovers plaintext message M by computing: (B^{-a}), δ mod p which can easily be proved as follows:

$$B^{-a} \cdot \delta \equiv g^{-ab} Mg^{ab} \equiv M(\bmod p)$$

Implementation of ElGamal encryption with artificially small parameters

Key generation: Entity A selects the prime p = 12884901893 and a generator g = 12 in Z^*_p with order p-1 = 12884901892.

- Alice (A) chooses the private-key: $a = 2^{10} = 1024 < p$.
- Computes: $A = g^a \bmod p = 12^{1024} (\bmod p) = 3505577916$
- A's public key ring is:
  (P, g, A) = (12884901893, 12, 3505577916).

Encryption (Bob): To encrypt a plaintext message M = 352247.

- Bob (B) selects private key, a random integer:
  $b = 2^{15} = 32768 < p$.
- Computes his public key as:
  $B = g^b = 12^{32768} \bmod p = 9663562615$.
- He computes the shared master secrete key using DH key exchange protocol as:

$$S_{AB} = (A)^b = (g^a)^b \equiv (3505577916)^{32768} \pmod p = 11093904324$$

- He encrypts ciphertext as:

  $\delta \equiv M \cdot S_{AB} \pmod p \equiv 553247.11093904324 \pmod p = 9930699416$

- B sends C = (B, δ) = (9663562615, 9930699416) to A.

Decryption (Alice): To recover the message M, Alice (A) does the following procedure:

- Receives ciphertext:
  C = (B, δ) = (9663562615, 9930699416)

- Computes master shared secret key:
  $S^{-1}_{AB} \equiv g^{-ab} \bmod p = 3314334791$

- Recovers M by computing:
  $M \equiv S^{-1}_{AB} \cdot \delta (\bmod p) \equiv 352247$
  where, we note that: $S^{-1}_{AB} \cdot \delta \equiv S^{-1}_{AB} \cdot M \cdot S_{AB} = M$

Common system-wide parameters: All entities may select to use the same prime p and generator g, in which case p and g need not be published as part of the public-key. This results in public-keys of smaller sizes. An additional advantage of having a fixed base g is that exponentiation can be expedited via precomputation. A potential disadvantage of common system-wide parameters is that larger modulo p may be warranted. For practical application and security reasons the crypto-keys must of 1024-bit or greater is recommended (Fig. 1).

DIGITAL SIGNATURE AND AUTHENTICATION

Authentication is more important than encryption[8,26-30]. Most people's security intuition says exactly the opposite, but it's true. Imagine a situation where Alice and Bob are using a secure communications channel to exchange data. Consider how much damage an eavesdropper (Eve) could do if she could read all the traffic. Then think about how much damage Eve could do if she could modify the data being exchanged. In most situations, modifying data is a devastating attack and does far more damage than merely reading it.

One way to address the authentication problem encountered in public-key cryptography is to attach digital signature to the end of each message that can be used to verify the sender of the message[7,28]. The significance of a digital signature is comparable to the significance of a handwritten signature. In some situations, a digital signature may be as legally binding as a handwritten signature. Once you have signed some data, it is difficult to deny doing so later-assuming that the private-key has not been compromised or out of the owner's control. This quality of digital signatures provides a high degree of nonrepudiation - i.e., digital signatures make it difficult for the signer to deny having signed the data[7,8,28]. This quality is stronger than mere authentication (where the recipient can verify that the message came from the sender); the recipient can convince a judge that the signer sent the message. To do so, he must convince the judge he did not forge the signed message himself! In authentication problem the recipient does not worry about this possibility, since he only wants to satisfy himself that the message came from the sender.

In short, an electronic signature must be a message-dependent, as well as signer-dependent. Otherwise the recipient could modify the message before showing the message-signature pair to the judge. Or he could attach the signature to any message whatsoever, since it is not possible to detect electronic cutting and pasting. To implement signatures the public-key cryptosystem must be implemented with trap-door one-way permutations i.e., have the property (d), since the decryption algorithm will be applied to unenciphered messages[7,8,10].

ElGamal was the first cryptographer to introduce the basic concept of digital signature scheme. The ElGamal signature scheme is similar to the encryption algorithm in that the public-key and private-key have the same form; however, encryption is not the same as signature verification, nor is decryption the same as signature creation as in RSA. The DSA is based in part on the ElGamal signature algorithm[7,8].

For a long period of time (1985-1996) after the birth of the ElGamal signature scheme and the family of such signatures (e.g., Schnorr and DSS), it was widely believed that the difficulty of factoring such a signature should somehow be related to the discrete logarithm in a large subgroup of a finite field[5,10,28]. However, no formal evidence (formal proof) was ever established until 1996. Poincheval and Stern succeeded in demonstrating affirmative evidence for relating the difficulty of signature forgery under a signature scheme in the ElGamal-family signatures to that of computing discrete logarithm[26]. They do so by making use of a powerful tool: The Random Oracle Model (ROM) for proof of security. The ROM-based technique of Pointcheval and Stern is an insightful instantiation of the general ROM-based security proof technique to proving security for the ElGamal-family signatures.

Digital Signature Algorithm: The Digital Signature Algorithm (DSA) was proposed in August 1991 by the US. National Institute of Standards and Technology (NIST) for use in their Digital Signature Standard (DSS) and was later specified in a US Government Federal Information Processing Standards (FIPS 186[29,30]) called the Digital Signature Standard (DSS). It was designed at the NSA as part of the Federal Government's attempt to control high security involving cryptography. Part of that policy included prohibition (with severe criminal penalties) of the export of high quality encryption algorithms. The DSS was intended to provide a way to use high security digital signatures across borders in a way which did not allow encryption. Those signatures required high security asymmetric key encryption algorithms, but the DSA (the algorithm at the heart of the DSS) was intended to allow one use of those algorithms, but not the other. It didn't work. DSA was discovered, shortly after its release, to be capable of encryption (prohibited high quality encryption, at that), however, it is so slow when used for encryption as to be even more than usually impractical.

The US government based their Digital Signature Algorithm (DSA) on much of ElGamal's work[5] and is the best known example of a large system where the Discrete Logarithm (DL) algorithm is used. Its security is based on the intractability of the Discrete Logarithm Problem (DLP) in prime-order subgroup of Z^*_p. As with the RSA algorithm, these transformations raise the computational complexity of the problem. The discrete logarithm system relies on the discrete logarithm problem modulo p for security and the speed of calculating the modular exponentiation for efficiency. In terms of computational difficulty, the discrete logarithm problem seems to be on a par with factoring[28].

The Mechanics of Digital Signature Algorithm (DSA): The Signature-Creation Data consists of the public parameter an integer y computed as: $y = g^x \bmod p$, as per the DLP above. Note that p and q are large prime numbers[31]. When computing a signature of a message M, no padding of the hashcode is necessary. However, the hashcode must be converted to an integer by applying the method described in Appendix 2.2[30].

The basic idea of DSA is for the signer of message M - that is, the possessor of the value x behind the publicly known, $g^x \bmod p$- to append a pair of numbers r and s obtained by secretly picking another number k between 1 and q, computing, $r = (g^k \bmod p)$, (i.e., computing $g^k \bmod p$) and then taking the remainder of that number mod p) and $s = k^{-1}$ (SHA(M)+xr) mod q where, k^{-1} is the multiplicative inverse of, k (mod q) and SHA is the Secure Hash Algorithm[8]. He then sends (M, r, s) to the communicating partner. Another NIST standard, SHA (official acronym is SHA-1) reduces a character string of any length to a 160-bit string of gibberish. In the implementation of DSA, q is a 160-bit prime divisor of p-1 and g is an element of order q in F^*_p.

The receiver of (M, r, s) from person g^x computes, $u = s^{-1}$ SHA(M) mod q and $v = s^{-1} r$ mod q and then checks that $((g^u)(g^x)^v$ mod q), equals r. If it doesn't, then, by elementary number theory, something definitely went wrong. If it does, then, according to NIST, you can safely assume that the message M came from the presumably unique individual who knows the discrete logarithm of g^x. Table 1 shows the sequence of DSA scheme.

Table 1: Digital Signature Algorithm (DSA)

Digital Signature Algorithm (DSA)
Key generation:

- Choose an L-bit prime p, where, $512 \leq L \leq 1024$ and is divisible by 64
- Choose a 160-bit prime q, such that, p-1 = qz where, z is any natural number
- Choose, h where $1<h<p-1$ such that $g = h^z \bmod p > 1$
- Choose x by some random method, where, $0<x<q$
- Compute $y = g^x \bmod p$
- Public key is (p, q, g, y). Private key is x

Note that (p, q, g) can be shared between different users of the system, if desired

Signing:

- Choose a random per message value k (called a nonce), where, $1<k<q$
- Compute $r = (g^k \bmod p) \bmod q$
- Compute $s = k^{-1} (H(m)+xr) \bmod q$, where, H(m) is the SHA-1 hash function applied to the message m
- Signature is (m, r, s)

Nonce means 'for the present time' or 'for a single occasion or purpose'
Verifying:

- Compute: $w = s^{-1} \pmod q$
- Compute: u1 = xH (m)(mod q)
- Compute: u2 = wr (mod q)
- Compute: $v = (g^{u1} * y^{u2} \bmod p) \bmod q$
- Signature valid if v = r

DSA is similar to ElGamal discrete logarithm cryptosystem signatures[5].

Implementation of Digital Signature Algorithm (DSA)

Here we will use an overly small prime and integer numbers to show how DSA can be implemented in real applications. In real practice, the DSA has the advantage that signatures are fairly short, consisting of two numbers of 160 bit (the magnitude of p). By comparison, the RSA signature is about three times long[2].

Key generation:

- Choose a prime number:
  p = 12884901893, which, gives q = 3221225473 and z = 4.
- Choose h = 246, such that $g = h^z \bmod p = 246^4 \bmod p = 3662186256$
- Choose x = 323 and compute:
  $y = g^x \bmod p = 3662186256^{323} \bmod p = 1727826790$
- Public key is: (p, q, g, y) = 12884901893, 3221225473, 3662186254, 1727826790. Private key is: x = 323.

Signing:

- Choose a random per message value k = 4822 (called a nonce), where, $1<k<q$
- Compute $r = (g^k \bmod p) \bmod q = 3564107243 \pmod q = 342881770$
- Choose m = H(m) = 434136 and compute $s = k^{-1} (H(m)+xr) \bmod q = 3180062802$
- Signature is (m, r, s) = (434136, 342881770, 3180062802)

Nonce means 'for the present time' or 'for a single occasion or purpose'

Verifying:

- Compute: $u1 = s^{-1} H(m) \pmod q = 738418804$
- Compute: $u2 = s^{-1} r \pmod q = 1144589831$
- Compute $v = (g^{u1} * y^{u2} \bmod p) \bmod q = 342881770$
- Signature valid, since: v = r = 342881770.

The security of DSA is based on the assumption that the only attacks are either those that work in the multiplicative subgroup of order q without exploiting any special properties of this group, or else by methods such as index-calculus ones, which work with group modulo p. There is no proof that some algebraic relations could not be exploited to find an improved algorithm.

To-date digital signature algorithm remains seemingly secure, until the methods of Shanks and Pollard's running times can be improved substantially or a more effective algorithm with better running time, to threaten its security. Such an alternative algorithm would not require a subexponential technique to break the DSA. A method that runs in time $q^{1/4}$ (group Q = <q> of prime order) would destroy it. Thus, the DSA seemed to have attained both a high level of security and low signature storage and implementation time. This lack of progress in developing better algorithm capable of breaking DSA has provided a subjective feeling of comfort to crypto-designers and led them to choose a crypto-security parameter close to the edge of what is feasible. However, recently the DSA has been superseded by the ECDSA, which is a similar system based on the group of an elliptic curve rather than a finite field[16].

DISCRETE LOGARITHM RELATED PROBLEM

Let G be a cyclic group and let g be generator of G such that: $G = \{g^0, g^1, \ldots, g^{n-1}\}$, where, n = |G| of G. The discrete log function $\mathrm{Dlog}_{g,G}$: G→Z takes input a group element γ and returns the unique $\alpha \in Z_n$ such that: $\gamma = g^\alpha$. There are several computational problems related to this function that are used in primitive. In all cases, we are considering an attacker that knows the group G and the generator g, that is, given g^α find α(DLP). Alternatively, given g^α and g^β, find $g^{\alpha\beta}$(DHP).

Methods for solving DLP: Recall communicating partners, Alice and Bob. We assume that they have decided in advance to use DH protocol, to form shared master secret key that they can use for secure communication, which is given by: $k = g^{s_A s_B} \pmod p$ where, $g^{s_A s_B} \in Z_p$ and $s = s_A s_B$ in range [1, n-1], where, n is the order of the group. Clearly, the key exchange systems, is broken as soon Eve, the benevolent eavesdropper, can determine s_A from the known k_A (or s_B from k_B). This is the motivation for us to look at various techniques to solve:

$$g^s \equiv r \pmod p \qquad (4)$$

where, g, r and p are known and s need to be determined. While the eavesdropper who happens to have overhead the exchange and thus knows g, g^{S_A} and g^{S_B}, will hopefully not be able compute g^S. The problem of how to solve, $g^s \equiv r \pmod p$, is called the discrete logarithm problem (DLP) (i.e., $\log_g r = s$)[32,33]. If the discrete log problem for the group G = <g>, order of group is easy, an eavesdropper can compute either s_A or s_B and can find out what g^s is. It is an important open question whether determining g^s knowing just g, g^{S_A} and g^{S_B} is as hard as the discrete log problem in general[34-36]. However, it is important to note that, a fast discrete log algorithm would definitely destroy the crypto-security and utility of the widely used Diffie-Hellman crypto-protocol. The same threat also affects other crypto-security based on DLP such as ElGamal cryptosystems and digital signature algorithm (DSA). This factor has generated huge research opportunity on the complexity of the discrete logs [37-39].

For real time practical application when implementing DLP-based crypto-algorithm like DH and the likes (where the certificate and signed hash have been added to prevent man-in-the-middle attack), the area of most concern focuses upon the fact that the specifications for generation of shared master secret key and certificate for authentication purposes, fixes the values of p and g. However, one must be careful since under some conditions, the discrete log is easy to compute and, for this reasons the value of p must be chosen carefully. For example, it is easy to compute the discrete logarithm when p-1 has only small prime factors, thereby susceptible to Pohlig-Hellman attack, which has time complexity bounded by the largest prime factor of the group. Therefore, for safe prime, p is usually chosen so that (p-1)/2 is itself prime, say q. Likewise, some care must be given to the choice of g so that the subgroup generated by g is relatively large, but this is usually an easier choice to make than the choice of p. The time complexity is thus same for both methods (one where the generator creates the whole group and the other where the generator generates only the subgroup with q)[40].

The best known generalized algorithms for solving discrete logarithms are still quite slow, but the majority of time is usually consumed in easily parallizable precomputations about the group (choice of p and g) in general. Once the precomputation is finished, then any discrete logarithm in that group is easily found. In that aspect and in most applications implementing secure network connections-it is known that most connections will be created using the only key exchange algorithm defined in the specification. Hence, the time and expense required to break the majority of secure connections is, therefore, only slightly greater than the time and expense required to perform the aforementioned precomputations for the group specified in that standard. One time-honored rule for security design is that the value of the data being protected should be less than the expense required to break the crypto-security systems.

GENERAL ATTACKS ON DISCRETE LOGARITHM PROBLEM

There are many ways Eve could implement to acquire the shared master secret key: one option is she could exploit the weakest link in the crypto-security systems. This could be via many available options, e.g., breaking the underlying crypto-algorithm and which in most cases is harder option to be attempted only as a last resort. Instead the eavesdropper might opt to exploit other weaknesses such as: recovering a key by observing the power consumption or electromagnetic radiation of the crypto-devices; finding vulnerabilities in the crypto-security protocols or simply revert to stealing the key: Through clever interactive social engineering with those trusted to safeguard the keys. In most cases, however, the crypto-algorithms are always the most important core tool in crypto-security applications.

If we assume that Eve's has no any other alternative available and so must resort to brute-force attack of the core crypto-algorithm, which in this case requires her to solve the underlying DLP, i.e.,

$$g^S \equiv r \pmod p \;\; (\text{or} \log_g r = s).$$

In applying brute-force approach to find, s, from g^s, she would have to try: s = 0, 1, 2, ... until a solution is found or, alternatively, to put: g^0, g^1, g^2, ... in a table and look for r. Either way, the complexity is p. If p consists of t bits, we can say that the complexity is given by 2^t, so the complexity grows exponentially in t. There are much better methods that she can resort to, which balance the time complexity with available memory, such as is the case with

Shank's Baby-Step Giant Step (BSGS) method. Other methods are Pollard's rho, Silver-Pohlig-Hellman and index calculus, which we all discuss in this paper. Of all this methods, the index calculus method, a subexponential-time algorithm, offer better performance in cracking DLP based crypto-schemes. In case of the prime fields, there is more advanced version of the index-calculus known as the Number Field Sieve (NFS), to solve the DLP with expected time[41]:

$$O(\exp(1.93 + o(1))(\ln q)^{1/3}(\ln \ln q)^{2/3})$$

If $q = 2^m$, there is a variation of the index-calculus known as Coppersmith's algorithm, to solve the DLP with expected time[38]:

$$O(\exp(c + o(1))(\ln q)^{1/3}(\ln \ln q)^{2/3})$$

for some c<1.587. Note that the DLP is still considered hard in these groups because the runtimes of these algorithms are not bounded by any polynomial in, q. However, the existence of these subexponential-time algorithms means that one must use larger key sizes than if only exponential-time attacks were known. For example, for practical security reasons it is recommended that prime fields, p should have at least 1024 bits (Fig. 1).

Shank's Baby-Step Giant-Step (BSGS): Suppose that one has enough memory available to store m elements of Z_p. Then the Shanks algorithm gives us an efficient method to balance the time complexity with the available memory to solve the discrete log: $g^s \equiv r \pmod{p}$. The method require one to compute $g^0, g^1, g^2, \ldots, g^{m-1}$, sort this element in a list to allow for easy look-up table, where incase of the baby steps (BS) the exponents increase by 1. Next check if the r is in the table, if not, then one checks if r/g^m is in the table, if not check for r/g^{2m} and continue, Giant Steps (GS). When r/g^{im} is in the table, say it equals $g^k, 0 \le k \le m-1,$ one has found the unknown exponent s = im+k. The time complexity of the baby-step method is p/m, so the product of memory requirement and time complexity is still $p \approx 2^t$.

Shank's Baby-Step Giant-Step (BSGS): System requirement: one has enough memory available to store m elements of Z_p.

Input: A finite group G = <g> of order n and g, y ∈ G.

Output: The discrete logarithm of y to the base g.

Procedure:

- Compute the ceiling of square root $n : m = \lceil \sqrt{n} \rceil$
- Construct a table T of (i, $g^{i.m}$) pairs, $1 \le i \le m$ and sort by the second component in the pair.
- Compute $y.g^j$, starting at j = 0 . (($y.g^j$) might be equal to g^{x+1} for some x and j).
- Compare $y \cdot g^j$ with $g^{i.m}$ entries in T. If a match is found then $g^{i.m} = g^{x+j}$ and thus $x = i \cdot m - j$, is the desired discrete log. If there is no such match, then try another value of j, with $j \le m$

Time complexity: $O(\sqrt{n})$ and Space complexity $O(\sqrt{n})$.

As an example, let $G = Z^*_{139}$. Then g = 2 is the generator of G with order of n = p-1 = 138. We look for the discrete logarithm of y = 43 to the base g.

Implementing BSGS: First let's compute $m = \lceil \sqrt{138} \rceil = 12$ which we use to construct table T of (i, $g^{i.m}$) pairs, for $1 \le i \le 12$

i	1	2	3	4	5	6	7	8	9	10	11	12
2^{12i} mod 139	65	55	100	106	79	131	36	116	34	125	63	64

Sorting T by the second entry:

i	9	7	2	11	12	1	5	3	4	8	10	6
2^{12i} (mod 139)	34	36	55	63	64	65	79	100	106	116	125	131

Computing $y \cdot g^j$ for j = 0, 1, 2,....

j	0	1	2	3	4	5
43.2^j (mod 139)	43	86	33	66	132	125

Comparing the two table entries, we can observe that we have a match in the entries for i = 10 and j = 5, which gives: $g^{12 \cdot (10)} = g^{5+x}$, i.e., the discrete logarithm of 43 to the base 2 in Z^*_{139} is x = 120-5 = 115. Hence, $\log_g y = \log_2 43 = 115$. Indeed: $2^{115} \equiv 43 (\mathrm{mod} 139)$.

In Shanks' algorithm, the checking for matching from the two sorted lists of m entries each can be done in linear time (assuming that the representations of elements are compact enough). Hence, the running time of the algorithm is dominated by the arithmetic required to compute the two (lists) tables and the time to sort them. Further, Shanks' algorithm is also known to be deterministic. Therefore, if one is willing to give up on determinism, one can replace sorting by hashing, thereby speeding up the process. On the other hand, there is no easy way to reduce space requirements (other than by increasing the running time), which are order m: $|<g>|^{1/2}$[42]. There are other general algorithms for discrete log problem that run in time $O(|<g>^{1/2}|)$ and very little space, both methods are randomized and are due to Pollard[13], which we discuss next.

Pollard's Rho (ρ) method for solving DLP: Basics Idea-Pollard's Rho algorithm is based on the birthday paradox[43,44]. If we randomly choose elements (with replacement) from a set of N numbered elements, we only need to choose about $\sqrt{N}$ elements until we get one element twice (called a collision). This can be applied to find discrete logarithms as follows. By choosing a, b $\in_R$ [0, N-1], one obtains a random group $g^a h^b$. Such group elements are randomly selected until we get a group element twice. If $g^{a_i} h^{b_i}$ and $g^{a_j} h^{b_j}$ represent the same group element then $a_i + b_i x \equiv a_j + b_j x \pmod{N}$, whence:

$$x = (a_j - a_i)(b_i - b_j)^{-1} \bmod N \qquad \text{for } b_i \not\equiv b_j \pmod{N} \quad (5)$$

The Pollard's rho method has an expected storage requirement given by $E(T) \approx \sqrt{\pi N/2}$, where, T is the random variable describing the number of group elements chosen until the first collision occurs. The main thrust of Pollard's rho method, is then how to detect collision without the need to store $\sqrt{\pi N/2}$ group elements. The collision in this method is done by means of a random function: $f: G \to G$ For actual implementations, f is chosen such that it approximates a random function as closely as possible.

The originally suggested function by Pollard (for Z^*_p) can be generalized towards arbitrary cyclic groups given by:

$$f(x,a,b) = \begin{cases} (hx, a, b+1) & \text{if } x \in S_1 \\ (x^2, 2a, 2b) & \text{if } x \in S_2 \\ (gx, a+1, b) & \text{if } x \in S_3 \end{cases} \quad (6)$$

where, S_1, S_2 and S_3 are three sets of roughly the same size which form a partition of G. However, Teske[39,40] has shown that the Pollard's function f is not random enough and gives alternative and better function as:

$$f(x) = x \cdot g^{m_s} h^{n_s}, \text{ if } x \in M_s \qquad \text{for } s \in \{1, 2, \ldots, r\} \text{ and } r \approx 20 \quad (7a)$$

where, again M_s are roughly of the same size and form a partition of G, which this time is partitioned into more than three subsets. For both functions, it is of course necessary that determining the subset M_i and S_i, respectively, to which a group element belongs is very efficient.

By starting at a random point $g^{a_0} h^{b_0}$ and iteratively applying a random function, random points $g^{a_i} h^{b_i}$ are generated. Because the group is finite, we eventually arrive at a point for the second time (i.e., a collision occurs), which happens after expectation E(T), thereby, the sequence of subsequent points then cycle forever. With very little time and space overhead, it is possible to detect such a cycle with Floyd's cycle-finding algorithm (or with an improved variant by Brent[45]).

The Mechanics of Pollard's Rho (ρ) method
Input: A finite group G = <g> of order n and g, y ∈ G.

Output: The discrete logarithm of y to the base g.

Procedure:

- Break the set G into three approximately equal-sized sets S_1, S_2 and S_3.
- Define a sequence of elements over G: $x_0, x_1, x_2, \ldots$ as: $x_0 = 1$

$$x_{i+1} = f(x_i) = \begin{cases} y \cdot x_i & \text{if } x_i \in S_1 \\ x_i^2 & \text{if } x_i \in S_2,\ i \ge 0 \\ g \cdot x_i & \text{if } x_i \in S_3 \end{cases} \quad (7b)$$

- This sequence, in turn, defines two other integers sequences a_i, b_i such that $x_i = g^{a_i} \cdot y^{b_i}$, for $i \ge 0$; $a_0 = b_0 = 0$

$$a_{i+1} = \begin{cases} a_i & \text{if } x_i \in S_1 \\ 2a_i & \text{if } x_i \in S_2,\ i \ge 0 \\ a_i + 1 & \text{if } x_i \in S_3 \end{cases} \quad (8)$$

$$b_{i+1} = \begin{cases} b_i + 1 & \text{if } x_i \in S_1 \\ 2b_i & \text{if } x_i \in S_2,\ i \ge 0 \\ b_i & \text{if } x_i \in S_3 \end{cases} \quad (9)$$

- **Cycle-finding calculation:** The next step is to find a pair (x_i, x_{2i}) with $x_i = x_{2i}$.

  In this case:

$$g^{a_i} \cdot y^{b_i} = g^{2a_i} \cdot y^{2b_i} \Rightarrow g^{2a_i - a_i} = y^{b_i - 2b_i}$$

$$\Rightarrow \log_g g^{2a_i - a_i} = \log_g y^{b_i - 2b_i} \Rightarrow \log_g y = \frac{2a_i - a_i}{b_i - 2b_i} \bmod n \quad (10)$$

As long as $b_i \not\equiv b_{2i} \bmod n$, the discrete log can be calculated as above. In the rare case that a collision $x_i = x_{2i}$ is not found, or that $b_i \equiv b_{2i} \bmod n$, the procedure can be repeated by selecting random $a_0, b_0 \in [1, n-1]$ and restarting with $x_0 = g^{a_0} \cdot y^{b_0}$.

Time-complexity: $O(\sqrt{n})$ Space-complexity: $O(1)$

Implementing Pollard's Rho for solving DLP: As an example, let H = Z^*_{383}. Then g = 2 is a generator of the subgroup G of Z^*_{383} of order n = 191. Suppose y = 228. Partitioning G into three sets according to the rule: $x \in S_1$ if $x \equiv 1 \bmod 3$; $x \in S_2$ if $x \equiv 0 \bmod 3$ and $x \in S_3$ if $x \equiv 2 \bmod 3$ and setting $x_0 = 0$; $a_0 = 0$ and $b_0 = 0$, results:

i	x_i	S_i	a_i	b_i	i	x_{2i}	a_{2i}	b_{2i}
1	228	0	0	1	2	279	0	2
2	279	0	0	2	4	184	1	4
3	92	2	0	4	6	14	1	6
4	184	1	1	4	8	256	2	7
5	205	1	1	5	10	304	3	8
6	14	2	1	6	12	121	6	18
7	28	1	2	6	14	144	12	38
8	256	1	2	7	16	235	48	152
9	152	2	2	8	18	72	48	154
10	304	1	3	8	20	14	96	118
11	372	0	3	9	22	256	97	119
12	121	1	6	18	24	304	98	120
13	12	0	6	19	26	121	5	51
14	**144**	0	12	38	**28**	**144**	10	104

The calculations show that: $x_{14} = x_{28} = 144$.

Computing:

$$\Delta a = 2a_{28} - a_{14} (\bmod n) = (10 - 12) \bmod 191 = 189$$
$$\Delta b = b_{14} - 2b_{28} (\bmod n) = (38 - 104) \bmod 191 = 125$$

Results:

$$x = \log_g y = \log_2 228 = (\Delta b)^{-1} \Delta a (\bmod n) =$$
$$(125)^{-1}(189) \bmod 191 = 110$$

Indeed:

$$2^{110} \equiv 228 (\bmod 383)$$

Pollard's Lambda (λ) method: Pollard's lambda method is known as Method for catching Kangaroos.

Input: A finite group G = <g> of order n; g, y ∈ G and value w standing for the size of an interval in which the discrete logarithm lies, e.g.,

$$A < \log_g y < B, w = B - A.$$

Output: The discrete logarithm of y to the base g.

Procedure:

- Compute two sequences T and W (called Kangaroo trails). The T sequence is $\{y'_0, y'_1, \ldots, y'_N\}$, where:

  $$y'_{i+1} = y_i \cdot g^{f(y'_i)}, \ i \geq 0$$

  Actually, this will occur if W's trail hits any point $y'_i, 0 \leq i \leq N$.
  T's trail begins at $y'_0 = g^B (\bmod n)$ and proceeds till y'_N. Note that

  $$y'_{i+1} = y_i \cdot g^{f(y'_i)} (\bmod n) \Rightarrow y'_N = y'_0 \cdot g^{d_N} (\bmod n)$$

  where, $d_i = \sum_{j=0}^{i-1} f(y'_j)$ is the distance from y'_0 till y'_i.

  W's trail begins at $y_0 = y = g^x$. The search ends when $y_M = y'_N$ for y_M in W's trail, at which point the discrete logarithm is calculated as:

  $$y_M = y'_N \quad \Rightarrow \quad y_0 \cdot g^{\sum_{i=0}^{M-1} f(y_i)} = y'_0 \cdot g^{\sum_{j=0}^{N-1} f(y'_j)}$$

  $$\Rightarrow \ y \cdot g^{d_M} = g^B \cdot g^{d_N} \ \Rightarrow \ g^{x+d_M} = g^{B+d_N} \ \Rightarrow$$
  $$x = (B + d_N - d_M) \bmod n$$

- If no collision (i.e., $y_M = y'_N$) occurs (the probability of which can be controlled) before d_M exceeds $B+d_N-A = w+d_N$, then the limit is terminated (failure). W has traveled beyond the trap. Subsequent iterations of the above sequences can be run as required.

Time-complexity: $O(\sqrt{w})$ and Space-complexity: O(log w). The Shanks method and the Pollard's kangaroo method can be used to compute the discrete log of y in about $\sqrt{m}$ steps when this discrete log is known to lie in an interval of length at most m. Hence, crypto-designers have to be careful not to limit the range in which discrete logs lie.

To-date the running times of Pollard and Shanks algorithms have not been improved to any substantial. This has since led to the assumption that in the absence of other structure in a cyclic group G = <g> of prime order, it will require on the order of $|G|^{1/2}$ operations to compute a discrete log in G. Many of the modern asymmetric key cryptosystems based on DLPs, such as DSA[2,8,9], rely on Schnorr method[10], which reduces the computational burden normally imposed by having to work in a large finite field by working within a large multiplicative subgroup Q = <q> of prime order. With an assumption that the discrete log in Q cannot be solved much faster than $\sqrt{q}$ steps. For q of order 2^{160}, as in DSA, this is about 10^{24} group operations. Since group operations are typically considerably more intensive than the basic instruction of ordinary computers ([46] for the case of ECC), it is reasonable to estimate that 10^{24} group operations might require at least 10^{26} ordinary computer instructions. A mips-year (MY) is equivalent to about $3{\cdot}10^{13}$ instructions, so breaking DSA, say, with Shanks or Pollard algorithms would require over 10^{12} MY, which appears to be adequate for a while at least. Several crypto-researcher, including Richard Crandall and Len Adleman, have observed that all the instructions executed by digital computers in history are on the order of Avogadro's number, about $6{\cdot}10^{23}$. The largest factoring projects so far have used 10^{17} operations and other large number distributed projects have accumulated on the order of 10^{20} operations (Table 2).

Table 2: Computing power available for integer factorization (in MY)

Year	Covert attack	Open project
2004	10^{8}	$2{\cdot}10^{9}$
2014	10^{10}-10^{11}	10^{11}-10^{13}

The Pohlig-Hellman Algorithm: Here we present the Pohlig-Hellman algorithm for computing discrete logarithms[47]. If G is not simple, it will reduce a DLP in G to several DLPs in smaller groups. It thus restricts the possible choices of groups in which the DLP is maximally hard. The idea is to take advantage of many-to-one homomorphic images in which the DLP is easier to solve. This will be important in the sequel because generalizations suggest that, even if one considers structures other than groups of the DLP, one should consider using simple structures. Specifically, using simple structure is the most reliable to avoid similar attacks[2,47,48].

The mechanics of Pohlig-Hellman Algorithm

Input: A finite group G = <g> of order n; g, y ∈ G.

Output: The discrete logarithm of y to the base g.

Procedure:

```
1. Let the factorization of n be $n = \prod_{i=1}^{r} p_i^{e_i}$, $e_i \geq 0$.
2. For i = 0 to r compute the decomposition
   $x_i = h_0 + h_1 \cdot p_i + \ldots + h_{e_{i-1}} \cdot p_i^{e_i - 1}$ where, $x_i = x(\mathrm{mod}\, p_i^{e_i})$
3. Set $\gamma = 1; h_{-1} = 0$ and $\bar{g} = g^{\frac{n}{p_i}}$.
4. Compute h_j for j = 0 to e_i-1 do
   compute
      $\gamma = \gamma \cdot g^{h_{j-1} \cdot p_i^{j-1}}$ and $\bar{y} = (y \cdot \gamma^{-1})^{n/p_i^{j+1}}$
   compute $h_j = \log_{\bar{g}} \bar{y}$
   (e.g., using Pollard's rho method)
   set $x_i = h_0 + h_1 \cdot p_i + h_2 \cdot p_i^2 + \ldots + h_{e_i - 1} \cdot p_i^{e_i - 1}$
5. Use the CRT to compute the discrete log x with
   $0 \leq x \leq n-1$ and $x \equiv x_i \bmod p_i^{e_i}$, with $1 \leq i \leq r$.
6. Output x.
```

Note that there are several well-known algorithms for performing step 5 in polynomial time. The idea is that in step 1 one can find 2, one can find $x_i = \log_g y(\mathrm{mod}\, p_i^{e_i})$. This is further reduced by finding the base-p expansion of x_i one digit at a time. Observe that each time step 4 is reached, $\bar{g}$ has order p_i. Thus, one need only compute discrete logarithm in subgroups of order p_i. Unless there is some trivial way to solve the DLP in G, this is more efficient than computing one discrete logarithm in the full group with order n. It is thus desirable that one should choose G with prime order so this algorithm yields no reduction at all.

In general, we observe that if G is not simple, then there exist groups G_i and homomorphisms of the form, $f_i : G \to G_i$ with trivial kernels. One may then solve the corresponding DLP in each homomorphic image. Furthermore, if there exist such G_i, such that:

$$f : G \to G_1 \times \cdots \times G_k$$

$$g \mapsto (f_1(g), \cdots, f_k(g))$$

is a monomorphism, then solving the DLP in each G_i solves the DLP in G up to an application of the Chinese remainder theorem.

Time complexity: $O(\sum_{i=1}^{r} e_i(\log n + \sqrt{p_i}))$, provided the factorization of n is given.

Space complexity: O(1) if used in conjunction with Pollard's rho method, for example.

Implementing Pohlig-Hellman Algorithm: Let G = Z^{*}_{271}, g = 43 a generator of Z^{*}_{271} of order n = 270. Let y = 210. The discrete logarithm is computed as follows:

- The prime factorization of n = 270 = $2{\cdot}3^{3}{\cdot}5$.

- (Compute $x_1 \equiv x \bmod 2$),
  Compute $\overline{g} = g^{n/2} \bmod p = g^{270/2} \bmod 271 = 270$
  and $\overline{y} = y^{n/2} \bmod p = y^{270/2} \bmod 271 = 270$
  Then $x_1 = \log_{270} 270 = 1$

- (Compute $x_3 \equiv x \bmod 5$),
  Compute $\overline{g} = g^{n/5} \bmod p = g^{270/5} \bmod 271 = 244$
  and $\overline{g} = g^{n/5} \bmod p = g^{270/5} \bmod 271 = 244$
  Then $x_3 = \log_{244} 187 = 2$

- Compute $x_2 \equiv x \bmod 3^3 = h_0 + h_1 \cdot 3 + h_2 \cdot 3^2$,
  compute: $\overline{g} = g^{n/3} \bmod p = g^{270/3} \bmod 271 = 28$
  compute:

  $$\gamma = 1 \text{ and } \overline{y} = (y \cdot \gamma^{-1})^{n/3} \bmod p = (y \cdot \gamma^{-1})^{270/3} \bmod 271 = 242,$$

  Then h_1 is found as: $h_0 = \log_{28} 242 = 2$

  compute: $\gamma = \gamma \cdot g^2 \bmod 271 = 223$ and $\overline{y} = (y \cdot \gamma^{-1})^{n/3^2} \bmod p = (y \cdot \gamma^{-1})^{270/9} \bmod 271 = 242,$

  Then h_1 is found as: $h_1 = \log_{28} 242 = 2$

  compute:

  $$\gamma = \gamma \cdot g^{2 \cdot 3} \bmod 271 = 223 \cdot 43^6 \bmod 271 = 68$$

  and

  $$\overline{y} = (y \cdot \gamma^{-1})^{n/3^3} \bmod p = (y \cdot \gamma^{-1})^{270/27} \bmod 271 = 1,$$

  Then h_2 is found as: $h_2 = \log_{28} 1 = 3$

  Hence:

  $$x_2 = h_0 + h_1 \cdot 3 + h_2 \cdot 3^2 = 2 + 2 \cdot 3 + 3 \cdot 3^2 = 35$$

- Solving the triple of congruences:

  $$x \equiv 1 (\bmod 2) \qquad x \equiv 2 (\bmod 5) \qquad x \equiv 35 (\bmod 27)$$

  we use CRT, which gives us the $x \equiv 197 (\bmod 270)$,
  such that $x = \log_{43} 210 = 197$
  Indeed: $43^{197} \equiv 210 (\bmod 271)$

INDEX-CALCULUS METHOD

Over finite fields where the DLP is defined, there is another additional structure beyond the multiplicative structure. The index-calculus methods take advantage of this extra structure[13,49].

The index calculus method is sub-exponential algorithm for solving the DLP over a finite group G, i.e., given g, y ∈ G, g a generator, we seek to find a value $\beta \in Z/(|G|)Z$ satisfying $y = g^{\beta}$. We say that $\beta = \log_g y$ or $\beta = \text{ind}_g\, y$, for the index of y in g.

The basic idea, which goes back to Kraitchik[50], is that if:

$$\prod_{i=1}^{m} x_i = \prod_{j=1}^{n} y_j \tag{11}$$

for some elements of GF(q)*, then:

$$\sum_{i=1}^{m} \log_g x_i \equiv \sum_{j=1}^{n} \log_g y_j (\bmod q - 1) \tag{12}$$

If we collect many equations of the above form (with at least one of them involving an element z such as g, for which $\log_g z$ is known) and they do involve many x_i and y_j, then the system can be solved. This is similar to the situation IFP[51]. Progress in index calculus algorithms has come from better ways of producing relations that lead to equations such as Eq. (11).

Index Calculus method involve forming relations in what is called a factor basis, $F = \{p_1, p_2, ..., p_1\}$, which is formed by choosing a list of the first ν primes from the order n of the finite group G. How many we choose is critical. The factor basis, F consists of all integers whose prime factors are all less than or equal to the largest prime on our list.

Once the number of relations found equals the number of prime factors of the elements in the factor basis we can solve the system of equations to recover the discrete logs of these primes. From these logs we can then, with more computation, recover the discrete log of any chosen element.

In determining the discrete logs of the elements, we start by looking at the exponentiations of the generator: g, g^2, g^3, ..., mapping these values to the integers if the field happens to be F_p instead of Z_p, which we can write in product form:

$$g^k = \prod_{i=1}^{\nu} p_i^{e_i},$$

where, $p_i \in F$ random integer k is such that $1 \le k \le p-1$. Compute: $g^k \in G$. If any value of g^k is in F, we record it and following relations. We can derive from g^k's factorization into powers of their first 1 primes and the fact that Z_p has order p-1.

$$k \equiv \sum_{i=1}^{v} e_i \cdot \log_g p_i (\bmod p-1) \qquad (13)$$

where, p_i is the ith prime from our factor base and e_i is its corresponding exponent in the factorization of g^k. We continue computing powers of the generator until we obtain v independent relations. We solve these equations (which are typically sparse) to get the discrete log of each of the v primes. Now, to find the discrete log of $y \in F_p$ we compute the quantities y, y.g, y.g^2, ... and lift these as well to Z if needed. We continue the computation until we find an element, y.g^β, that factors completely using our factor basis such that:

$$y \cdot g^\beta = \prod_{i=1}^{k} p_i^{e_i},$$

and taking logs on both sides, gives:

$$\beta + \log_g y \equiv \sum_{i=1}^{r} e_i \cdot \log_g p_i (\bmod p-1) \qquad (14)$$

where, e_i is its corresponding exponent to the ith prime. We already know the discrete log of each of the v primes, so we can just solve this equivalence for, $\log_g y$. Here, we get a runtime that is subexponential in p provided we make a good choice for $r \approx e^{\sqrt{\log p}}$ [52].

How can we subvert this attack? We can use really large keys as previously discussed, but this has significant drawbacks. The system becomes even slower with larger keys making it undesirable. Additionally, large keys require more computation space so such a cryptosystems cannot fit on small form, constrained environment wireless devices that also need to utilize encryption e.g., smartcards. Since larger keys do not seem to offer much of a fix, we need to consider other alternative cryptosystems like ECC which make use of ECDLP instead of the classical DLP, for more detail on ECDLP[18-20].

The mechanics of index-calculus
Input: A finite G = <g> of order n and g, y ∈ G

Output: The discrete logarithm of y to base g

Procedure:

- (Factor Base) find a subset F = {p_1, p_2, ⋯, p_1} of G such that a significant fraction of all elements in G can be efficiently expressed as a product of elements from F.
- (Linear Relations)
  - Select a random integer k, 0≤k≤n-1 and compute g^k.
  - Try to write g^k as a product of elements from F:

$$g^k = \prod_{i=1}^{v} p_i^{e_i}, \ e_i > 0 \qquad (15)$$

  - If Eq. 15 is successful, then taking the discrete log of both sides of Eq. 15 results in the linear relations:

$$k = \sum_{i=1}^{v} e_i \cdot \log_g p_i \bmod n \qquad (16)$$

  - Collect (t+c) linear relations like Eq. 16, by choosing other random k's

- (Discrete Logs of elements F) working modulo n, solve the linear system of (v+c) equations in v unknowns and obtain the value of the discrete logarithms of the factor base: $\log_g p_i$, $1 \le i \le v$.
- (Actual computation of the discrete log) select a random r, 0≤r≤n-1 and compute y.g^r.
- Try to write y.g^r as a product of elements of F:

$$y \cdot g^r = \prod_{j=1}^{v} p_j^{e_j}, \ e_j > 0 \qquad (17)$$

- If y.g^r is not repsentable as (17) then choose another r and retry the factorization (17). Otherwise, taking discrete logs of both sides of (3) results:

$$\log_g y = \sum_{i=1}^{v} e_i \cdot \log_g p_i - r = x \qquad (18)$$

Time-complexity:

- $L_p(\frac{1}{3}, 1.923)$ for G = Z^*_p using NFS-variant of index calculus.
- $L_{2^m}(\frac{1}{3}, c)$ for G = GF(2^m) and c<1.587

Implementation of index-calculus method: To help us under the power of cracking DLP based cryptosystems, we will use Index Calculus to recover the master shared secret key we found using DH key exchange crypto-protocol. Recall too, that Alice and Bob used the same key to exchange secure data using ElGamal crypto-algorithm. Suppose Eve at her non-descriptive hideout seeks to recover the key, so she needs to solve:

$$g^s \equiv 11093904324 (\bmod p)$$

where, $s(= s_A s_B)$ is the shared key, $g = 12$ is the generator and $p = 12884901893$ is the prime integer.

We start with a precomputation of the right-hand side, i.e., 11093904324. We consider our factor base which consist of the first 15 prime numbers:

$$F = \{p_1, p_2, ..., p_{15}\}$$
$$= \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47\}$$

and try to solve the logarithm problem for the all elements in the factor base. In other words, we want to solve:

$$g^{k_1} \equiv 2 \pmod p,\ g^{k_2} \equiv 3 \pmod p,\ \cdots,\ g^{k_{15}} \equiv 47 \pmod p$$

Now select a random exponent integer s and compute $12^s \pmod p$ and check if the residue can be factored completely by means of the factor base. For example:

$$12^{422} \equiv 12084561537 \equiv 3 \cdot 17 \cdot 23 \cdot 10302269 \pmod p$$

where, we observe that 10302269 cannot be factored further over our factor base F. We also recall that the larger the factor base, the easier we find the residue that do completely factor with respect to the factor base, but the price we pay is having the more unknowns k_i completely factor with respect to our factor base, we end up with linear relation between the unknown k_i's, e.g.,:

$$12^{9625853812} = 4616675 = 5^2 \cdot 7 \cdot 23 \cdot 31 \cdot 37 \pmod p$$

gives the relation:

$$9102048310 = 2 \cdot k_1 + k_2 + k_3 + k_4 + k_5 + k_6 + k_7 \pmod{p-1}$$

where, each k_i's is given by, $k_i = \log_g p_i \pmod p$.

Next we collect enough relations to enable us solve the unknown k_i's, e.g.,:

$$12^{9102048310} = 78540 = 2^2 \cdot 3 \cdot 5 \cdot 11 \cdot 17 \pmod p$$

$$12^{7382258392} = 254646 = 2 \cdot 3^2 \cdot 7 \cdot 43 \cdot 47 \bmod p)$$

$$12^{12298253855} = 26455 = 5 \cdot 11 \cdot 13 \cdot 37 \pmod p$$

$$12^{4835947410} = 119510 = 2 \cdot 5 \cdot 17 \cdot 19 \cdot 37 \pmod p$$

$$12^{6525201043} = 4238785 = 5 \cdot 23 \cdot 29 \cdot 31 \cdot 41 \pmod p$$

where, we get the following linear relations:

$$9102048310 = 2k_1+k_2+k_3+k_4+k_5+k_7 \pmod{p-1}$$
$$7382258392 = k_1+2 \cdot k_2+k_{14}+k_{15} \pmod{p-1}$$
$$12298253855 = k_3+k_5+k_6+k_{12} \pmod{p-1}$$
$$4835947410 = k_1+k_3+k_7+k_8+k_{12} \pmod{p-1}$$
$$6525201043 = k_3+k_9+k_{10}+k_{11}+k_{13} \pmod{p-1}$$

The solutions are given as follows:

$$k_1 = \log_{12} 2 = 420700703$$
$$k_2 = \log_{12} 3 = 4470896487$$
$$k_3 = \log_{12} 5 = 3803513117$$
$$k_4 = \log_{12} 7 = 2365150781$$
$$k_5 = \log_{12} 11 = 3116054641$$
$$k_6 = \log_{12} 13 = 3492178431$$
$$k_7 = \log_{12} 17 = 12702231662$$
$$k_8 = \log_{12} 19 = 8006496046$$
$$k_9 = \log_{12} 23 = 1050306206$$
$$k_{10} = \log_{12} 29 = 127183706$$
$$k_{11} = \log_{12} 31 = 9601764817$$
$$k_{12} = \log_{12} 37 = 1886507666$$
$$k_{13} = \log_{12} 41 = 4827335089$$
$$k_{14} = \log_{12} 43 = 7844296260$$
$$k_{15} = \log_{12} 47 = 9793819458$$

where, all k_i's are computed modulo p-1.

Finally, we are ready to solve our original problem: $12^s \equiv 11093904324 \pmod p$. Pick a random exponent s and combining with y and then check if, $y \cdot g^s \pmod p$, completely factors over our factor base. After a few tries we get lucky, such that:

$$y.g^s \equiv 110993904324 \cdot 12^{8766128316} \pmod p \equiv 19050076 = 2^2 \cdot 3^3 \cdot 11^2 \cdot 31 \cdot 47$$

$$\log_g y + s \equiv 2k_1+3k_2+k_5+k_{11}+k_{15} \pmod{p-1}$$

$$\log_g y \equiv 2k_1+3k_2+2k_5+k_{11}+k_{15} - s \pmod{p-1} \equiv (2 \cdot 4207002703+3 \cdot 4470896487+96017648457+ 9793819456-8766128312 \equiv 33554432 \pmod{p-1}$$

$$\Rightarrow \quad \log_g y = \log_{12} 110930904324 = 33554432$$

Indeed: $12^{33554432} \equiv 11093904324 \pmod p$

Time complexity of the Index Calculus is given by, $\exp(1.923t^{1/3} (\text{in } t)^{2/3})$[53]. The memory requirement typically equals the square root of this the time complexity. This

Table 3: The complexity of different methods to take discrete logarithms for $p \approx 2^t$

Algorithms	Time	Memory
Exhaustive key search	2^t	1
Baby-step giant-step	$2^t/m$	m
Pollard	$2^t/2$	1
Index calculus	$e^{1.923 t^{1/3} (\ln t)^{2/3}}$	$e^{1.923 t^{1/3} (\ln t)^{2/3}}/2$

makes the index calculus algorithm and variants, the only method for taking discrete logarithms with subexponential complexity, Table 3.

THE FUTURE STATE OF THE ART IN IFP AND DISCRETE LOGARITHM COMPUTATION

The basic question at the onset for implementer of the DLP-based crypto-algorithms is how large discrete log problems that can be handled by current available state of the art computational tools? For a conservative estimate, it is appropriate to warn that to obtain a proper estimate of discrete log problems it is better to consider what has been accomplished in Integer Factorization Problem (IFP). Much more effort has been devoted to IFP than to discrete logs and most of the leading algorithms are similar. Thus, although discrete logs in prime fields do appear harder than factoring integers of the same size, it is prudent to disregard this difference when choosing crypto-security implementation. Number Field Sieve (NFS) is currently at the cutting-edge of research into integer factoring algorithm capable of factoring large composite numbers over 100 digits[54]. The current record in factoring a generally hard integer is that of the 200 decimal digits challenge integer from RSA Data Security, Inc., RSA-200, which was accomplished with general number field sieve (GNFS) was factored on May 9, 2005 by Bahr, Boehm, Franke and Kleinjung[55]. Among the Cunningham integers, the record is the factorization of 248 decimal digit integer by Special Number Field Sieve (SNFS) was factored by Aoki, Kida, Shimoyama, Sonoda and Ueda (CRYPTREC) on April 04, 2004[56,57].

Computing power is measured in MIPS-years: a million-instructions-per-second computer running for one year or about 3×10^{13} instructions. A 100-MHz Pentium III is about a 50-MIPS machine; a 1600-node Intel Paragon is about 50,000 MIPS. In 1983, a Cray X-MP supercomputer factored a 71-digit number in 0.1 MIPS-years, using 9.5 CPU hours. That's expensive. Factoring the 129-digit number in 1994 required 5000 MIPS-years and used the idle time on 1600 computers around the world over an eight-month period. Although it took longer, it was essentially free. These two computations used what's called the quadratic sieve , but a newer, more powerful algorithm has arrived. The general number field sieve is faster than the quadratic sieve for numbers well below 116 digits and can factor a 512-bit number over 10 times faster-it would take less than a year to run on an 1800-node Intel Paragon. Figure 2 for current security level involving MIPS-years estimation featuring public keys.

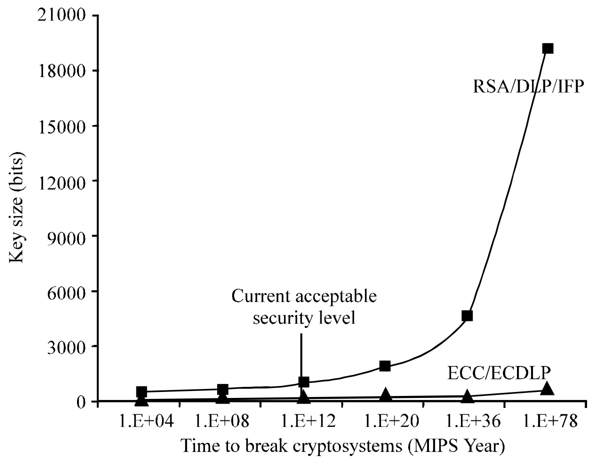

Fig. 2: Comparison of security levels of ECC and RSA/DSA

Odlyzkol in paper[58] argues that, given the record of improvements in the index calculus algorithms, it seems imprudent to assume that the current version of GNFS is the best that will be available for a long time. At the least, it seems a reasonable precaution to assume that future algorithms will be as efficient as today's SNFS, in which even 1024 bit RSA moduli might be insecure for anything but short-term protection.

Therefore, the baseline and trade-off, is the size of n should be chosen such that the time and cost for performing the factorization exceeds the value of the secured/encrypted information. But even then, great care must still be taken in the overall crypto-design, as current development in integer factorization have gone much faster than foreseen and it is a precarious matter to venture upon quantitative forecasts in this field. Moreover, one should realize that it always remains possible that a new computational method could be invented from unsuspecting quarter, which makes factoring easy (e.g., quantum computing, if an operative quantum computer were to be realized in the not-so distance future)-fortunately or unfortunately depending on which side you are on-no one knows how to build one yet!

Shor[59,60] showed that if such machine could be built, integer factorization and discrete logs (including elliptic curve discrete logs) could be computed in polynomial time. This result has stimulated an explosion in research on quantum computers[61]. The one comforting factor is that all experts agree that even if quantum computers are eventually built, it will take many years to do so (at least for machines on a scale that will threaten modern public key cryptosystems) and so there will be advance warning about the need to develop and deploy alternative crypto-algorithms.

CONCLUSIONS

We have discussed a method for implementing a public-key cryptosystem whose security rests in part on the difficulty of solving discrete logs. If the crypto-security designs and methods are appropriately implemented, it permits secure communications to be established without the use of courier to carry keys and it also permits one to sign digitized documents.

In general, the strength of encryption is related to the difficulty of discovering the key, which in turn depends on both the cipher used and the length of the key. No matter which technique you choose, you must keep in mind that a desperate cryptanalyst can always decipher the message. Hence, you should always take all the necessary precautions to protect your data. Those precautions range from proper choice of cryptographic keys to physically protecting your assets and yourself.

In overall the baseline and trade-off, is the size of n and/or prime p should be chosen such that the time and cost for cracking the crypto-systems exceeds the value of the secured/encrypted information. But even then, great care must still be taken in the overall crypto-design, as current development in integer factorization and equivalently solving DLP, have gone much faster than foreseen and it is a precarious matter for one to venture upon quantitative forecasts in this field. Moreover, one should realize that it always remains possible that a new computational method could be invented from unsuspecting quarter, which makes factoring 'easy' (e.g., quantum computing). Today, the wise crypto-designer is ultraconservative when choosing key lengths for a public key cryptosystems. He must consider the intended security, the key's expected lifetime and the current state of art in factoring or equivalently solving DLP.

REFERENCES

1. Odlyzko, A., 1984. Discrete logarithms in finite fields and their cryptographic significance. In Advances in Cryptology Eurocrypt'84, Springer-Verlagm pp: 224-314.
2. Menezes, A., P. Van Oorschot and S. Vanstone, 1997. Handbook of Applied Cryptography. CRC Press.
3. Diffie, W. and M.E. Hellman, 1966. New directions in cryptography. IEEE Transaction on Information Theory, IT-22: 644-654.
4. Diffie, W. and M.E. Hellman, 1976. Multi-user cryptographic techniques. Proceedings of AFIPS Natl. Comput. Conf., pp: 109-112.
5. ElGamal, T., 1985. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inform. Theory, 31: 469-472.
6. Anonymous, 1986. National Institute of Standards and Technology. Digital Signature Standard, FIPS PUB.
7. Rabah, K., 2004. Data security and cryptographic techniques. A review. Pak. J. Inform. Technol., 3: 106-132.
8. Rabah, K., 1993. Secure Implementing message digest. Authentication and Digital Signature, (In Press).
9. Schneier, B., 1993. Applied Cryptography. John Wiley and Sons New York.
10. Schnorr, C., 1991. Efficient signature generation by smart cards. J. Cryptol., 4: 161-174.
11. Nyberg, K. and Rueppel, 1996. Message recover for signature schemes based on the discrete logarithm problem Designs, Codes and Cryptography, 7: 61-81.
12. Pollard, J.M., 1978. Monte Carlo methods for index computations. Math. Comput., 32: 918-924.
13. Lenstra, A.K., H.W. Lenstra, Jr., M.S. Manasse and J.M. Pollard, 1990. The number field sieve, Proceeding 22nd ACM Symp. Theory of Computing, pp: 564-572.
14. Rivest, R., A. Shamir and L. Adleman, 1978. A method for obtaining digital signatures and public key cryptosystems. Comm. ACM., 21: 120-126.
15. Lenstra, A.K. and E.R. Verheul, 1999. Selecting Cryptographic Key Sizes. J. Cryptol.

16. Rabah, K., 2005. Theory and implementation of elliptic curve cryptography. J. Applied Sci., 5: 604-633.
17. Miller, S., 1986. Use of elliptic curves in cryptography. In CRYPTO '85, pp: 417-426.
18. Koblitz, N., 1987. Elliptic curve cryptosystems, Mathematics of Computation, 48: 203-209.
19. Certicom^{TM}, Securing Innovation, http://www.certi com.com/index.php.
20. Rabah, K., 0000. Theory and Implementation of Data Encryption Standard: A Review. (In Press).
21. Merkle, R.C., 1978. Secure communication over insecure channels. Communications of the ACM., 21: 294-299.
22. Knuth, D.E., 1969. The Art of Computer Programming. Vol. 2, Seminumerical Algorithms Addison-Wesley, Reading, Mass., 2: 00-00.
23. Pollard, J.M. and Monte Carlo, 1978. Methods for index computation (mod p). Mathematics of Computation, 32: 918-924.
24. Maurer, U.M., 1994. Towards the equivalence of breaking the diffie-hellman protocol and computing discrete logarithms. In Advances in cryptology, CRYPTO '94, pp: 271-281.
25. Menezes, A.J. and S.A. Vanstone, 1992. A note on cyclic groups, finite fields and the discrete logarithm problem. AAECC, 3: 67-74.
26. Pointcheval, D. and J. Stern, 1996. Security proofs for signature schemes. In Eurocrypt'96, LNCS 1070, pp: 387-398.
27. Pointcheval, D. and J. Stern, 2000. Security arguments for digital signatures and blind signatures. J. Cryptol., 13: 361-396.
28. Rabin, M.O., 1979. Digital signature and public-key functions as intractable as factorization. MIT Laboratory of Computer Science, Technical Report, MIT/LCS/TR-212.
29. Anonymous, 1993. Digital signature standard. National Institute of Standards and Technology. FIPS PUB., 186.
30. Anonymous, 2000. Digital Signature Standard (DSS). National Institute of Standards and Technology, NIST: FIPS Publication, pp: 186-192.
31. Rivest, R.L., 1991. Finding four Million Random Primes. In: Advances in Cryptology-Crypto'90. Menezes, A.J. (Ed.,), LNCS 537, Springer-Verlag, pp: 625-626.
32. Koblitz, N., 1998. Algebraic Aspects of Cryptography. Springer-Verlag, Berlin. With an appendix by A.J. Menezes, Y.H. Wu and R.J. Zuccherato (Eds.).
33. Eisenbud, D., C. Huneke and W. Vasconcelos, 1992. Direct Methods for Primary Decomposition. Inventiones Mathematicae, 110: 207-235.
34. Maurer, U. and S. Wolf, 1998. Lower Bounds on Generic Algorithms in Groups. In: Advances in Cryptology-EUROCRYPT '98, (K. Nyberg, Ed.), LNCS No. 1403, pp: 72-84.
35. Boneh, D. and R. Venkatesan, 1996. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Advances in Cryptology -CRYPTO '96, (N. Koblitz, Ed.), Lecture Notes in Computer Science No. 1109, pp: 129-142.
36. Hastad, J. and M. Naslund, 1998. The security of individual RSA bits. In: Proceeding 39th Found. Comp. Sci. Symp., IEEE, pp: 510-519.
37. Odlyzko, A., 1985. Discrete Logarithms in Finite Fields and Their cryptographic Significance. Advances in Cryptology: Proceedings of EUROCRYPT 84, Beth, T., N. Cot and I. Ingemarsson (Eds.), Springer-Verlag, LNCS No. 209, pp: 224-314.
38. Odlyzko, A., 1994. Discrete Logarithms and Smooth Polynomials, Finite Fields: Theory, Applications and Algorithms, (Mullen, G.L. and P. Shiue, Eds.), Am. Math. Soc., Contemporary Math., No. 168, pp: 269-278.
39. Schirokauer, O., D. Weber and T. Denny, 1996. Discrete Logarithms: the Effectiveness of the Index Calculus Method. In Algorithmic Number Theory: Second Intl. Symp., ANTS-II, (Cohen, H., Ed.), Lecture Notes in Math. No. 1122, pp: 337-362.
40. Van Oorschot, P.C. and M.J. Wiener, 1996. On diffie-hellman key agreement with short exponents. In: Advances in Cryptology EUROCRYPT'96, LNCS 1070, pp: 332-343.
41. Gordon, D.M., 1993. Discrete logarithms in GF(p) using the number field sieve, SIAM J. Discrete Math., 6: 124-138.
42. Amirazizi, H.R. and M.E. Hellman, 1988. Time-memory-processor trade-offs, IEEE Trans. Inform. Theory, 34: 505-512.
43. Teske, E., 1998. Speeding up Pollard's Rho Method for Computing Discrete Logarithms. In: Proceedings of ANTS III-The 3rd Intl. Symp. Algorithmic Number Theory, (J.P. Buhler, Ed.), LNCS, 1423: 351-357.
44. Teske, E., 2001. On Random Walks for Pollard's Rho Method. Mathematics of Computation, 70: 809-825.
45. Brent, R.P., 1980. An Improved Monte Carlo Factorization Algorithm, J-BIT, 20: 176-184.

46. Escot, A.E., J.C. Sager, A.P.L. Selkirk and D. Tsapakidis, 1998. Attacking elliptic curve cryptosystems using the parallel Pollard Rho method. CryptoBytes (The technical newsletter of RSA Laboratories), 4: 15-19.
47. Pohlig, S.C. and M.E. Hellman, 1978. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Trans. Inform. Theory, 24: 106-110.
48. Pfister, G., W. Decker and G. Greuel. In: Matzat, B., G. Greuel and G. Hiss, Eds., Algorithmic Algebra and Number Theory, pp: 187-220.
49. Coppersmith, D., 1984. Fast evaluation of logarithms in fields of characteristic two. IEEE Trans. Inform. Theory, 30: 587-594.
50. McCurley, K.S., 1990. The Discrete Logarithm Problem. In: Cryptography and Computational Number Theory, (Pomerance, C., Ed.), Proceeding Symp. Applied Math. Am. Math. Soc., 42: 49-74.
51. Lenstra, A.K. and H.W. Lenstra, Jr., 1993. The development of the number field sieve. Lecture Notes in Mathematics No. 1554.
52. Harasawa, R., J. Shikata, J. Suzuki and H. Imai, 1999. Comparing the MOV and FR Reductions in Elliptic Curve Cryptography. EUROCRYPT, pp: 190-205.
53. Bach, E. and J. Shallit, 1996. Algorithmic Number Theory. The MIT Press, Cambridge, 1: 00-00.
54. Cavallar, S., W. Lioen, H. te Riele, B. Dodson, A. Lenstra, P. Leyland, P. Montgomery, B. Murphy and P. Zimmermann, 1999. Factorization of RSA-140 using the number field sieve, Asiacrypt, pp: 195-207.
55. Bahr, Boehm, Franke and Kleinjung, Record Integer Factoring- RSA-200.
56. A 248-digit factorization using SNFS, http://www.rkmath.rikkyo.ac.jp/~kida/snfs248e.htm.
57. Victor Miller's number theory mailing list archive, available at _ http://www.listserv.nodak.edu
58. Odlyzko, A.M., 1995. The future of integer factorization. CryptoBytes (The technical newsletter of RSA Laboratories), 1: 5-12.
59. Shor, P.W., 1994. Algorithms for Quantum Computation. Discrete logarithms and factoring. In: Proceedings of the 35th Ann. Symp. Found. Comp. Sci., (Ed. Goldwasser S.), (IEEE Comp. Soc. Press, Los Alamitos, CA), pp: 124-134.
60. Shor, P.W., 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Computing, 26: 1484-1509.
61. Quantum Physics e-print archive, http://xxx.lanl.gov/archive/quant-ph.