Differentiated access based on cryptographic methods

Gheorghe Grigoraş, Dana Dănciulescu and Nicolae Constantinescu

***Abstract*—** Starting from the basic identity-encryption, were proposed by the other authors. The models were assumed in the random oracle model with a variant of the computational Diffie-Hellman problem. In our paper we propose a new variant of Identity-Based Encryption (IBE), using elliptic curves schemes and prove his security completeness. Also, we are doing a complete description of the necessary system used in present scheme in order to secure a communication. The performance of our system is comparable to the performance of ElGamal encryption in F*p and the security of the system is based on the elliptic curve calculus intractability.

***Keywords*—** Identity-Based Encryption, Diffie-Hellman problem, Elliptic Curves Cryptography, Escrow ElGamal.

I. Introduction

There have developed security algorithms of information based on making the Public Key System which has an equivalent in an identity system [6]. Initially, this system was created in order to assure the confidentiality of the communication between the members in a group who knows identity information of that who want to communicate. This is, for example, the email address, the surname, the first name, a code on the identity card, etc. So, this kind of scheme has:

Setup: generate the control parameters in order to identify the users and create the master key. Extract: it uses the master key in order to generate the private key corresponding to public key that was generated in function of the Identity String (IS). Encrypt: the description of the encryption algorithm that will be applied on the plain text. It will use the public key. Decrypt: the decryption algorithm of the ciphered message. It will use the corresponding private key.

Many of other proposed schemes use an important quantity of processor-time [4, 8, 7, 5] to calculate the private key, or they consider a dedicated hardware as being preexistent.

To describe our model in this work, we'll first illustrate a general model of the scheme. This model is based on the creation of the public key which is not necessary to be taken from a server. By convention, it considers the public key is made by the concatenation of the complete name of the receiver and the current week. So, we obtain IS (Identity String). It will create an algorithm which transforms some kind of string in a key and this key will become the encryption public key used to encrypt the plain text and it will be sent by communication channel.

The data transferred from one system to another over public network can be protected by the method of encryption. On encryption the data is encrypted/scrambled by any encryption algorithm using the 'key'. Only the user having the access to the same 'key' can decrypt/de-scramble the encrypted data. This method is known as private key or symmetric key cryptography. There are several standard symmetric key algorithms defined. Examples are AES, 3DES etc. These standard symmetric algorithms defined are proven to be highly secured and time tested. But the problem with these algorithms is the key exchange. The communicating parties require a shared secret, 'key', to be exchanged between them to have a secured communication. The security of the symmetric key algorithm depends on the secrecy of the key. Keys are typically hundreds of bits in length, depending on the algorithm used. Since there may be number of intermediate points between the communicating parties through which the data passes, these keys cannot exchange online in a secured manner. In a large network, where there are hundreds of system connected, offline key exchange seems too difficult and even unrealistic. This is where public key cryptography comes to help. Using public key algorithm a shared secret can be established online between communicating parties without the need for exchanging any secret data.

In public key cryptography each user or the device taking part in the communication have a pair of keys, a public key and a private key, and a set of operations associated with the keys to do the cryptographic operations. Only the particular user/device knows the private key whereas the public key is distributed to all users/devices taking part in the communication. Since the knowledge of public key does not compromise the security of the algorithms, it can be easily exchanged online.

A shared secret can be established between two communicating parties online by exchanging only public keys and public constants if any. Any third party, who has access only to the exchanged public information, will not be able to calculate the shared secret unless it has access to the private key of any of the communicating parties.

The receiver will solicit the private key from the Public Key Generator (PKG). This private key is corresponding with the public key that is characteristic to this user (receiver). The decryption will be produced with this key, so Public Key

Manuscript received November 2011.

Generator holds the control of all the decryption keys and the public keys associated with every user. To communicate with Public Key Generator, any participant has a secret code that is also known by the PKG. It is used to assure the Challenge-Response algorithm in order to make the secure communication between PKG and any participant.

A. One-way function

In public key cryptography, keys and messages are expressed numerically and the operations are expressed mathematically. The private and public key of a device is related by the mathematical function called the one-way function. One-way functions are mathematical functions in which the forward operation can be done easily but the reverse operation is so difficult that it is practically impossible. In public key cryptography the public key is calculated using private key on the forward operation of the one-way function. Obtaining of private key from the public key is a reverse operation. If the reverse operation can be done easily, that is if the private key is obtained from the public key and other public data, then the public key algorithm for the particular key is cracked. The reverse operation gets difficult as the key size increases. The public key algorithms operate on sufficiently large numbers to make the reverse operation practically impossible and thus make the system secure. For e.g. RSA algorithm operates on large numbers of thousands of bits long.

B. Key Agreement

Key agreement is a method in which the device communicating in the network establishes a shared secret between them without exchanging any secret data. In this method the devices that need to establish shared secret between them exchange their public keys. Both the devices on receiving the other device's public key perform key generation operation using its private key to obtain the shared secret.

As we see in the previous section the public keys are generated using private key and other shared constants. Let P be the private key of a device and $U(P, C)$ be the public key. Since public key is generated using private key, the representation $U(P, C)$ shows that the public key contain the components of private key P and some constants C where C is known by all the device taking part in the communication.

Consider two devices A and B. Let P_A and $U_A(P_A, C)$ be the private key and public key of device A, and P_B and $U_B(P_B, C)$ be the private key and public key of device B respectively. Both device exchanges their public keys.

Device A, having got the public key of B, uses its private key to calculate shared secret:

$K_A=Generate\_Key(P_A, U_B(P_B, C))$

Device B, having got the public key of A, uses its private key to calculate the shared secret

$K_B=Generate\_Key(P_B, U_A(P_A, C))$

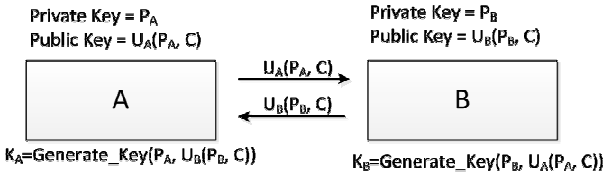

Figure 1. Key Agreement

The key generation algorithm 'Generate_Key' will be such that the generated keys at the device A and B will be the same, that is shared secret $K_A=K_B=K(P_A, P_B, C)$.

Since it is practically impossible to obtain private key from the public key any middleman, having access only to the public keys $U_A(P_A, C)$ and $U_B(P_B, C)$, will never be able to obtain the shared secret K. Examples of key agreement algorithms are DH, RSA and ECDH.

During the key exchange process the public keys may pass through different intermediate points. Any middleman can thus tamper or change the public keys to its public key. Therefore for establishing shared secret it is important that device A receives the correct public key from device B and vice versa.

II The RSA Algorithm

One of the biggest problems in cryptography is the distribution of keys. Suppose you live in the United States and want to pass information secretly to your friend in Europe. If you truly want to keep the information secret, you need to agree on some sort of key that you and he can use to encode/decode messages. But you don't want to keep using the same key, or you will make it easier and easier for others to crack your cipher. But it's also a pain to get keys to your friend. If you mail them, they might be stolen. If you send them cryptographically, and someone has broken your code, that person will also have the next key. If you have to go to Europe regularly to hand-deliver the next key, that is also expensive. If you hire some courier to deliver the new key, you have to trust the courier, et cetera.

A Trap-Door Ciphers

But imagine the following situation. Suppose you have a special method of encoding and decoding that is "one way" in a sense. Imagine that the encoding is easy to do, but decoding is very difficult. Then anyone in the world can encode a message, but only one person can decode it. Such methods exist, and they are called "one way ciphers" or "trap door ciphers".

Here's how they work. For each cipher, there is a key for encoding and a different key for decoding. If you know the key for decoding, it is very easy to make the key for encoding, but it is almost impossible to do the opposite—to start with the encoding key and work out the decoding key. So to communicate with your friend in Europe, each of you has a trap door cipher. You make up a decoding key D_a and generate the corresponding encoding key E_a. Your friend does exactly the same thing, but he makes up a decoding key D_b and

generates the corresponding encoding key E_b. You tell him E_a (but not D_a) and he tells you E_b (but not D_b). Then you can send him messages by encoding using E_b (which only he can decode) and vice versa—he encodes messages to you using E_a (which only you can decode, since you're the only person with access to D_a). Now if you want to change to a new key, it is no big problem. Just make up new pairs and exchange the encoding keys. If the encoding keys are stolen, it's not a big deal. The person who steals them can only encode messages—they can't decode them. In fact, the encoding keys (sometimes called "public keys") could just be published in a well-known location.

B Certification

There is, of course, a problem with the scheme above. Since the public keys are really public, anyone can "forge" a message to you. So your enemy can pretend to be your friend and send you a message just like your friend can—they both have access to the public key. Your enemy's information can completely mislead you. So how can you be certain that a message that says it is from your friend is really from your friend?

Here is one way to do it, assuming that you both have the public and private keys E_a, E_b, D_a, and D_b as discussed in the previous section. Suppose I wish to send my friend a message that only he can read, but in such a way that he is certain that the message is from me. Here's how to do it. I will take my name, and pretend that it is an encoded message, and decode it using D_a. I am the only person who can do this, since I am the only person who knows D_a. Then I include that text in the real message I wish to send, and I encode the whole mess using E_b, which only my friend knows how to decode.

When he receives it, he will decode it using D_b, and he will have a message with an additional piece of what looks to him like junk characters. The junk characters are what I got by "decoding" my name. So he simply encodes the junk using my public key E_a and makes certain that it is my name. Since I am the only one who knows how to make text that will encode to my name, he knows the message is from me. You can encode any text for certification, and in fact, you should probably change it with each message, but it's easy to do.

C RSA Encryption

Previously we described what a trap-door cipher means, further on we will see how one is made. The most used cipher of this form is "RSA Encryption" which will be described in this chapter. I will find two huge prime numbers, p and q that have 100 or maybe 200 digits each. I will keep those two numbers secret (they are my private key), and I will multiply them together to make a number $N = pq$. That number N is basically my public key. It is relatively easy for me to get N; I just need to multiply my two numbers. But if you know N, it is basically impossible for you to find p and q. To get them, you need to factor N, which seems to be an incredibly difficult problem.

But exactly how is N used to encode a message, and how are p and q used to decode it? Below is presented a complete example, but I will use tiny prime numbers so it is easy to follow the arithmetic. In a real RSA encryption system, keep in mind that the prime numbers are huge.

In the following example, suppose that person A wants to make a public key, and that person B wants to use that key to send A a message. In this example, we will suppose that the message A sends to B is just a number. We assume that A and B have agreed on a method to encode text as numbers. Here are the steps:

1. Person A selects two prime numbers. We will use $p = 23$ and $q = 41$ for this example, but keep in mind that the real numbers person A should use should be much larger.

2. Person A multiplies p and q together to get $pq = (23)(41) = 943$. 943 is the "public key", which he tells to person B (and to the rest of the world, if he wishes).

3. Person A also chooses another number e which must be relatively prime to $(p-1)(q-1)$. In this case, $(p-1)(q-1) = (22)(40) = 880$, so $e = 7$ is fine. e is also part of the public key, so B also is told the value of e.

4. Now B knows enough to encode a message to A. Suppose, for this example, that the message is the number $M = 35$.

5. B calculates the value of $C = Me(\text{mod } N) = 357(\text{mod } 943)$.

6. $35^7 = 64339296875$ and $64339296875(\text{mod } 943) = 545$. The number 545 is the encoding that B sends to A.

7. Now A wants to decode 545. To do so, he needs to find a number d such that

$ed = 1(\text{mod } (p-1)(q-1))$, or in this case, such that $7d = 1(\text{mod } 880)$. A solution is $d = 503$, since $7*503 = 3521 = 4(880) + 1 = 1(\text{mod } 880)$.

8. To find the decoding, A must calculate $C^d(\text{mod } N) = 545^{503}(\text{mod } 943)$. This looks like it will be a horrible calculation, and at first it seems like it is, but notice that

$503 = 256+128+64+32+16+4+2+1$ (this is just the binary expansion of 503). So this means that:

$545^{503} = 545^{256+128+64+32+16+4+2+1} = 545^{256}545^{128} \ldots 545^1$

But since we only care about the result $(\text{mod } 943)$, we can calculate all the partial results in that modulus, and by repeated squaring of 545, we can get all the exponents that are powers of 2. For example, $545^2(\text{mod } 943) = 545 \cdot 545 = 297025(\text{mod } 943) = 923$. Then square again: $545^4(\text{mod } 943) =$

$(545^2)^2(\text{mod } 943) = 923 \cdot 923 = 851929(\text{mod } 943) = 400$, and so on. We obtain the following table:

$545^1(\text{mod } 943) = 545$
$545^2(\text{mod } 943) = 923$
$545^4(\text{mod } 943) = 400$
$545^8(\text{mod } 943) = 633$
$545^{16}(\text{mod } 943) = 857$
$545^{32}(\text{mod } 943) = 795$
$545^{64}(\text{mod } 943) = 215$
$545^{128}(\text{mod } 943) = 18$

$545^{256}(mod\ 943) = 324$

So the result we want is:

$545^{503}(mod943)=324 \cdot 18 \cdot 215 \cdot 795 \cdot 857 \cdot 400 \cdot 923 \cdot 545(mod\ 943) = 35.$

Using this tedious (but simple for a computer) calculation, *A* can decode *B*'s message and obtain the original message $N = 35$.

D RSA Key Generation

The RSA algorithm [10] has gained widespread use in the software industry and it is utilized by more and more people each year. RSA key pairs are the basis for ensuring both the privacy of data in RSA ciphertexts as well as the non-repudiability of digitally signed messages. However, the security of these basic functionalities rests on the honest and correct generation of RSA key pairs.

There are many subtle security issues surrounding the generation and use of RSA keys. For example, there are instances in which a malicious user may deliberately try to generate and certify a weak public key. The user can choose the prime p in the RSA public key $n = pq$ such that $p-1$ is smooth (a smooth integer has no large prime divisors). This allows anyone to factor n using Pollard's $p-1$ factoring algorithm [11]. The notion of generating a weak key may appear counter-intuitive to many readers. Why would anyone ever want to do a thing like that? The reasons for doing it are many. Perhaps the biggest reason for doing so is to be able to back out of a contract if business begins to turn south. By certifying a "weak" RSA public key, the signer will be in a position down the road to repudiate any and all digital signatures that were created using the corresponding private key. This can be argued on mathematical grounds in a court of law since a weak key is a key that can be factored by anyone. Hence, everyone has the ability to produce signatures using the weak key pair. Weak keys are particularly attractive to a malicious user when the existence of the weakness is not readily apparent, since in this case another user is not likely to produce forgeries under the malicious user's name.

However, to avoid being blamed for deliberately generating a weak key, the signer would have to convince a disinterested third party (such as a judge) that the use of the weak key was not deliberate. This is a challenge since it must be proven that a trustworthy key generation algorithm was used and that it was not tampered with. When RSA keys are generated randomly, the probability that a key is weak is already very small, and the use of *strong primes* reduces the risk even more. Strong primes have certain properties that make the product n hard to factor by specific factoring methods. Such properties include the existence of a large prime factor of $p-1$ and a large prime factor of $p+1$. This is one of the issues in the strong primes debate. Why else would a user want to certify a weak key?

Consider the possibility of political insurgency. A person from country *A* starts working for the government in country *B* and acts under cover. The person certifies a weak public key and uses it to store, receive, and transmit (in key exchange protocols) highly sensitive information. This has the potential to severely damage country *B*. Also, an "innocent" weakness in the key could get the person off the hook if he or she is accused. Of course, a simpler approach to the problem is for a malicious user to simply publish his or her private key in some inconspicuous fashion. The user would later point out the location of the private key and state that anyone could have obtained it. However, this argument is not likely to hold up in court. One would have to assess the probability that the private key would show up naturally, without the intervention of the key owner, and the chances of this are very small indeed. In truth weak keys are not likely to be generated, even when the simplest methods for generating RSA keys is used. However, it is possible using simple RSA key generation. So, a malicious user can try to make the case that, e.g., $p-1$ just happened to be smooth. These issues illustrate the importance of being able to validate RSA keys [12]. Even an honest user may generate and use an easily factorable RSA public key without realizing it. This happens when a malicious insider, such as the programmer that creates the RSA key generation device, inserts a backdoor that lets the insider obtain the user's private key. The reasons why a programmer would want to insert such a backdoor are obvious. It would permit the programmer to gain illicit access to encrypted information such as e-mails, secure socket connections, and so forth, and would also allow the programmer to impersonate users (e.g., forging signatures of other users, accessing the accounts of other users, etc.).

The scope of the problem is by no means specific to RSA. Backdoor attacks have been shown to exist in Diffie-Hellman, ElGamal, DSA, elliptic curve cryptosystems, and more. The scope of the problem is not limited to insider attacks either. An outsider is often in a position to insert a backdoor as well. Malicious software such as viruses and worms can insert a backdoor as part of their payload. The scope of this problem is immense, especially considering the fact that a backdoor can often be exploited in a completely covert way. For instance, when the attacker simply reads information but does not modify information it is often difficult to detect that the attack is even occurring. A tamper-resistant microchip is an ideal medium for planting a backdoor, since by its very nature the backdoor is well-hidden. Even when a key generation algorithm is implemented in software, the program is effectively a "black-box" in the eyes of the average user, since a deep understanding of mathematics as well as the underlying assembly language is necessary to discern the true nature of the program.

E Proving the Form of n=pq

When an RSA key generation device outputs two RSA primes, the key owner can perform rather simple tests on the correctness of the outputs. For instance, the key owner can

check that p and q are primes, that they are the correct size, etc. However, for ensuring nonrepudiability it is necessary that other users be convinced as well that $n = pq$ was generated properly. The purpose is to convince others that n is not "weak" and that there is no backdoor in use. This is a challenging problem since it is often the case that only the key holder is allowed to know p and q. Certain properties relating to the correctness of RSA key generation can be verified simply be performing computations on n. For instance, it can be publicly verified that: n is not prime, n is not divisible by small primes, and that n is not a perfect power (see for sieving algorithms). Performing these checks is a good measure, since a weak key such as $n=p^2q^2$ will be readily discovered. These verifications help to show that n was properly generated, but they are not sufficient. For example, when a 1024-bit key n is generated these verifications will not detect the case that p is 160 bits in length and q is 864 bits in length. One way that a black-box key generation implementation can prove these more complex assertions regarding n is to utilize non-interactive zero-knowledge proof systems. In a nutshell, a non-interactive zero-knowledge proof system consists of a proof generating algorithm and a corresponding verification algorithm.

The proof generating algorithm proves some type of assertion regarding a problem instance. For instance, the problem instance may be an integer n and the assertion may be that n is the product of two distinct prime numbers. The output of the proof generating algorithm is a data file that is typically anywhere from 40 kilobytes to over a 100 kilobytes in size. The data file has the property that it reveals nothing about the secrets associated with the problem instance (e.g., it does not expose p or q). The verification algorithm takes this file and the problem instance as input and verifies whether or not the file is valid. A valid file implies that the assertion holds with overwhelming probability. Several such proof systems can be utilized to prove the form of n. These proof systems are the subject of this section. The following are some well-known zero-knowledge interactive protocols that show various properties of n. All of these protocols can be converted into non-interactive zero-knowledge proof systems. Peralta and van de Graaf presented a

protocol that proves in perfect zero-knowledge that n is a Blum integer [13]. They define the set of Blum integers to be integers of the form $n=p^rq^s$ where $p, q \equiv 3 \bmod 4$, r and s are odd, and p and q are prime. A zero-knowledge protocol has been given that proves that n is square-free. Recall that an integer n is said to be square-free if m^2 does not evenly divide n for any $m > 1$. The protocol utilizes a parameter k that signifies the number of rounds in the protocol. By making k large enough, a cheating prover has a negligible chance of convincing the verifier that n is square-free when in fact it is not. A protocol that proves that n is a Blum integer combined with a protocol that proves that n is square-free proves that n is contained in the subset of Blum integers characterized by $r = s = 1$. Zero-knowledge protocols have been developed that prove surprisingly complicated facts about n. For instance, a statistical limited-knowledge protocol (that leaks very little information) has been given that proves that n is the product of two primes that are nearly equal in size, assuming that n has already been proven to be the product of two distinct primes [14]. A statistical zero-knowledge protocol has been given that proves that n is the product of two quasi-safe primes [15]. Finally, there is a zero-knowledge protocol that proves that n is the product of two safe primes. This last protocol is asymptotically efficient but would be cumbersome to utilize in practice.

An interesting open question regarding Blum integers is the following. Is there a probabilistic (or deterministic) algorithm for deciding whether or not n is a Blum integer? The existence of such a predicate would eliminate the need for a zero-knowledge proof that n is a Blum integer. These algorithms and protocols that prove various properties of n are helpful since they prove that there are no obvious weaknesses in the structure of n. However, they do not sufficiently protect against various forms of abuse. There is a wealth of literature surrounding the abuse of key generation algorithms, digital signature algorithms, and so on. These abuses are the subject of the next section.

F Cryptographic Abuses of RSA Key Generation

Gus Simmons initiated the investigation of abuses that involve clandestine information leakage within the context of cryptographic algorithms and protocols. The classic problem that demonstrates this type of abuse is known as the Prisoner's Problem. In the prisoner's problem, two prisoners are allowed to communicate to each other but are not allowed to send encrypted messages to each other. They are only permitted to exchange public keys and digitally sign their messages. The problem is to devise a way, using the digital signature algorithm in question, for the two prisoners to communicate secretly with each other through digital signatures in such a way that the warden cannot detect or read the subliminal messages. Such a communications channel is called a subliminal channel.

Yvo Desmedt noted that a subliminal channel exists in composites, and that use of this channel constitutes an abuse of RSA key generation. One way to implement a subliminal channel in composites is as follows. A subliminal message m_s and a checksum t of m_s are concatenated together (denoted by $m_s \parallel t$). The resulting string is asymmetrically encrypted using the public key of the recipient of m_s. The asymmetric cryptosystem must be probabilistic to ensure that c is pseudorandom. Let c be the resulting ciphertext. A random prime p and a random pad RND are chosen.

The quotient q and remainder r are then solved for in $c \parallel RND = pq + r$. If q is composite then this process is repeated. When q is prime, the public key is n = pq = (c || RND)−r. At worst a borrow bit will be taken from c, but this can be rectified. The subliminal message m_s is recovered by decrypting c and $c+1$ and verifying which of the two

checksums is correct. Note that the security parameter for c is half of the value of the security parameter for n. This approach is based on kleptography [16][17].

The subliminal channel in composites can be used to let one prisoner communicate secretly with another (although they have to keep generating new keys to keep communicating this way). Therefore, typical RSA key generation is subject to information leakage abuse by inmates. RSA key generation is also subject to abuse by malicious insiders. Consider the following rather simple attack. The attacker, who is the programmer that is creating the RSA key generation algorithm, stores a secret seed in the key generation algorithm and the algorithm supplies this seed to pseudorandom number generator. The fact that the seed is chosen uniformly at random and is "secure" leads to a cryptographically secure pseudorandom bit sequence. This sequence is known to the attacker and can be the sole source of randomness for deriving output pairs (p, q). The attack amounts to replacing the "honest" random sequence that is inherent to a probabilistic Turing machine with a "dishonest" pseudorandom sequence that is completely reconstructable by the insider. An RSA key pair that is compromised in this way allows the insider to read anything encrypted using the user's public key, and allows the insider to forge any signed document on behalf of the user. This type of attack is a very general one, since it can be applied to any probabilistic algorithm, not just cryptographic ones. Research has been conducted to investigate insider attacks against cryptographic algorithms with the specific goal of making the attacks robust from the attacker's perspective. This type of attack is far more attractive to an attacker than generating a "weak" key that can be exploited by anyone. The goal in this type of attack is to plant a backdoor in the key generation algorithm that: (1) generates keys that are indistinguishable from "normal" keys, (2) is robust against reverse-engineering, and (3) generates keys that are cryptographically secure with respect to everyone except the attacker. This type of attack gives the attacker an exclusive advantage.

It has been shown how to use the notion of a subliminal channel to mount this type of attack against RSA key generation. The attack makes use of the subliminal channel in composites $n = pq$ where n is a W-bit quantity. The intuition behind the insider attack is as follows. If there were a way to display randomly generated information in the bit representation of $n = pq$ such that: (1) only the insider can access the information, (2) only the insider can detect that the information is there, and (3) the information allows the insider to factor n, then a robust attack against RSA key generation would exist. The fact that the information is randomly generated each time that a key pair is generated provides security going forward with respect to a passive reverse-engineer.

A heuristic version of this attack is as follows.

It makes use of a pseudorandom number generator (PRNG) denoted by G. The insider places his or her own public key in the device. This key is used to compute c in the subliminal channel. The device chooses m_s randomly. It then computes the pseudorandom bit sequence $G(m_s)$. The bits in this sequence are considered $W/2$ bits at a time. The first such sequence that is a $W/2$-bit prime becomes p. If p leads to a prime value for the quotient q in the channel, then $n = (c \,||\, RND)-1$ is output as the user's public key. The insider obtains this public key from a CA, for instance. The insider then uses his or her own private key to obtain m_s. Given m_s it is then straightforward to recover p and factor n. The attack is robust against reverse-engineering since only the public key of the insider, not the corresponding private key, is revealed upon inspecting the RSA key generation code. Furthermore, compromised composites are computationally indistinguishable from uncompromised composites under reasonable intractability assumptions, thus assuring that no one will ever know that the attack is being carried out [16][17].

This type of attack is called a secretly embedded trapdoor with universal protection (SETUP). The attacker's public key is the secretly embedded trapdoor. The advantage of a SETUP attack over using a fixed pseudorandom bit sequence is that it provides secrecy going forward. That is, even if the key generation device is reverse-engineered and its state is revealed, it will not help the reverse-engineer factor the future (or even past) RSA keys that are produced. This is because the seed m_s is chosen randomly each time that the key generation algorithm is invoked. This is of particular importance in software implementations in which each user obtains the exact same copy of the key generation software. These types of attacks are by no means unique to RSA key generation. In addition they have been shown to exist in discrete-logarithm based cryptosystems. A SETUP attack has been shown against the Diffie-Hellman key exchange that leaks one of the two Diffie-Hellman exponents. An attack has been shown against the Digital Signature Algorithm that leaks the private key, and other attacks have been shown as well.

G Curbing Abuses of Cryptosystems

The standard approach to mitigating insider abuse in a cryptographic algorithm is to prevent the algorithm from having the luxury of controlling the final randomness that is used to derive the output values. This is enforced by requiring the use of a protocol to perform the computation, as opposed to a stand-alone algorithm. Gus Simmons introduced the idea of using randomization to destroy subliminal channels. To destroy a particular subliminal channel that was identified, Simmons has the warden generate a random number $x \in R_n$ (the ring of residues modulo n) and modify the message that was being sent from one prisoner to the other prisoner using x. For other early results that use randomization to eliminate subliminal channels.

To address the problem of key generation abuse, Desmedt investigated abuse-free ways of generating key pairs. His protocol is outlined within the context of the prisoner's problem, but applies to other abuses as well such as the

previously mentioned attack that uses a pseudorandom number generator. In this protocol an inmate Alice and a warden jointly generate a key pair such that only Alice knows the private key and such that the warden is convinced that no form of abuse is occurring.

A high-level description of this protocol is as follows. Alice commits to a random string r_a and the warden sends Alice a random string r_w. Alice performs the bitwise exclusive-or operation to obtain the random string $r = r_a \oplus r_w$. If r does not satisfy the properties needed to make the key pair (e.g., it does not lead to two RSA primes) then Alice reveals r_a to the warden. If r does lead to a proper public key, then Alice proves this in zero-knowledge. The overhead of this zero-knowledge protocol is substantial in practice since it relies on a very general zero-knowledge interactive protocol construction.

III A new proposed encryption scheme

In this part of article we describe the modality of secure information that will be transferred to another participant trough the communication channel. This scheme has four parts:

Setup

Every participant of the group has a control key, a personal parameter noted ID, known by himself and by the Public Key Generator. It is used to assure the secure communication between him and PKG.

Extract

Using the personal parameter ID and a message key it will communicate the private key for the participant. The private key will be generated by the Public Key Generator at the request of a participant authenticated by an ID.

Encryption

Another participant, using the transformation algorithm of a string (IS) in a public key $pk \in \{0,1\}^*$ will encrypt the plain text with this key and the cipher text such obtained will be transferred by the communication channel.

Decryption

Using the private key, obtained from the Public Key Generator, one participant will decrypt a message that is destined for him.

As we have already discussed, the algorithm used to obtain the private key from the Public Key Generator is a Challenge-Response type. Therefore, for its construction we'll use, in the same way like in [3, 2] the following phases:

Definition

The expression x^3+1 is a permutation of F_p, where E is an elliptic curve given by the equation $y^2 = x^3 + 1$, defined over F_p, and p is a prime number which satisfy $p = 6q - 1$, where q is a prime number, $q > 3$.

Let be $P \in E / F_p$ a generator in a group of points of order $q = (p+1)/6$.

Choice

Let be a point $y_0 \in F_p$, there is an unique point x_0, so that $(x_0, y_0) \in E / F_p$

Transformation

Let be $1+\zeta \in F_{p^2}$ a solution of the equation $x^3 - 1 = 0 \bmod p$. A transformation equation $\phi(x, y) = (\zeta x, y)$ is an automorphism of groups on the elliptic curve E. If $P = (x, y) \in E / F_p$ then $\phi(P) \in E / F_{p^2}$, but $\phi(P) \notin E / F_p$. Determination

The points $P, \phi(P)$ generate an isomorphic group $Z_q \times Z_q$. We note this group of points as being $E[q]$. These points can be calculated like in [3, 2].

Transformation IS-PK

The public key which will encrypt the plain text is made from a string obtained through the concatenation between the complete name of the receiver and the current weak. As we have a string, it is necessary to transform it in a key, PK. For this, at the beginning, we code the string (in accordance with *ASCII*) to obtain $IS^{(1)} \in \{0,1\}^*$. Using a cryptographic function, denoted here *H*, with $H : \{0,1\}^* \to F_p$., we'll construct the point *Q*: y_0 will be equal with $H(IS^{(1)})$ and $x_0 = (y_0^2 - 1)^{1/3} = (y_0^2 - 1)^{(2p-1)/3} \bmod p$. So that, $Q = (x_0, y_0) \in E / F_p$, and we'll compute $Q_{IS^{(1)}} = 6Q$ in order to obtain the necessary order for *Q* (this is denoted in [2] as being a random oracle model).

A Key life and the way to have the key in all parts

As we pointed out before, IS is made by the concatenation of the complete name of a participant and the current weak. This means that the key will be changed in every weak. Therefore, the validity of a pair of the key (public key and private key) is maintained only a weak.

B The functional scheme

In this part of the article we will describe the four steps of the algorithm which assures the confidentiality of the communications.

Setup

As we described before, we chose an elliptic curve *E* generated through a prime number *p, p= 6q-1*, a prime number *q, q>3*. We chose $P \in E / F_p$ according with the

transformation IS-PK. Also, we select $s \in Z_q^*$ and $P_{pub} = sP$. The master key is $s \in Z_q^*$.

Extract

As we have $IS^{(1)} \in \{0,1\}^*$, the Public Key Generator will construct a private key, hereby: $H(IS^{(1)}) = IS^{(2)} \in \{0,1\}^*$. From this point it results that $Q_{IS^{(2)}} \in E / F_p$ of order *q*. The private key will be $d_{IS^{(2)}} = sQ_{IS^{(2)}}$, where s is the master key.

Encrypt

Let be T the plain text to be encrypted by a sender. The ciphered text C will be obtained as being $(rP, T \oplus H(Q^r))$, where $Q = (Q_{IS^{(2)}}, P_{pub}) \in F_{p^2}$ and *r* is chosen with random oracle, $r \in Z_q$.

Decrypt

We have $C = (U, V)$ a ciphered text with the public key $Q_{IS^{(2)}}$. We'll obtain again the plain text from the ciphered text as follows: $T = V + H(R)$, where $R = (d_{IS^{(2)}}, U) \in F_{p^2}$.

C Security

The security of the system is given by the security of public keys system based on elliptic curves. These notions are in details described and prove in [1,9]. The mode of creation of the public key and private key doesn't offer advantages in the determinations of plain text for any attacker.

IV Conclusions and future works

In this paper, the authors proposed a new encrypt scheme of IBE type based on elliptic curves. This scheme represents a powerful method to assure the confidentiality of communications in a group which has a PKG. The advantages are given by the facility of the generation of the private and the public keys and by the fact that an user A which wants to send a message to any other participant B, doesn't allow to establish a link with PKG, in order to obtain the public key of B. It is enough to know the name of B. Also, only PKG knows about the key system of every user and the code allocated to everyone.

As a first application, we want to implement this scheme in a network with 163 systems. The next step consist of the creation of an hierarchy systems in the network, in order to have access to the information, so that any user is able to read all the messages that circulate in the tree, between the users who are hierarchies situated in theirs descendant nodes of the tree

REFERENCES

[1] 1) I.F. Blake, G. Seroussi, N.P. Smart, "Elliptic Curves in Cryptography", 2000 Cambridge University Press

[2] 2) Dan Boneh, Matt Franklin, "Identity-Based Encryption from the Weil Pairing", CRYPTO 2001, LNCS 2139, pp. 213-229, 2001 Springer-Verlag Berlin Heidelberg

[3] 3) D. Boneh, M. Franklin, "Identity based encryption", Full version available at http://crypto.stanford.edu/ibe

[4] 4) Y. Desmedt and J. Quisquater, "Public-key systems based on the di_culty of tampering", Proc. Crypto '86, pp. 111-117, 1986.

[5] 5) U. Maurer and Y. Yacobi, "Non-interactive public-key cryptography", proc. Eurocrypt '91, pp. 498-507.

[6] 6) A. Shamir, "Identity-based cryptosystems and signature schemes", Proc. Crypto '84, pp. 47-53.

[7] 7) S. Tsuji and T. Itoh, "An ID-based cryptosystem based on the discrete logarithm problem", IEEE Journal on Selected Areas in Communication, vol. 7, no. 4, pp. 467-473, 1989.

[8] 8) H. Tanaka, "A realization scheme for the identity-based cryptosystem", Proc.Crypto '87, pp. 341-349, 1987.

[9] 9) E. Verheul, "Evidence that XTR is more secure than supersingular elliptic curve cryptosystems", Proc. Eurocrypt 2001.

[10] 10) R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures

[11] and Public-Key Cryptosystems. In Communications of the ACM, vol. 21, no. 2, pages 120–126, 1978.

[12] 11) J. M. Pollard. Theorems on Factorizationand Primality Testing. In Proceedings of the Cambridge Philosophical Society, vol. 76, pages 521–528, 1974.

[13] 12) R. D. Silverman. RSA Public Key Validation. In Workshop on Cryptography and Computational Number Theory, K. Y. Lam, I. Shparlinski, H. Wang, C. Xing (Eds.), Progress in Computer Science and Applied Logic, vol. 20, Birkh¨auser, 2001.

[14] 13) J. van de Graaf, R. Peralta. A simple and secure way to show the validity of your public key. In Advances in Cryptology—Crypto '87, C. Pomerance (Ed.), LNCS 293, pages 128–134, Springer-Verlag, 1987.

[15] 14) M. Liskov, R. Silverman. A Statistical Limited-Knowledge Proof for Secure RSA Keys. Submitted to IEEE P1363 working group.

[16] 15) R. Gennaro, D. Micciancio, T. Rabin. An Efficient Non-Interactive Statistical Zero- Knowledge Proof System for Quasi-Safe Prime Products. In Conference on Computer and Communications Security, pages 67–72, ACM, 1998.

[17] 16) A. Young, M. Yung. The Dark Side of Black-Box Cryptography, or: Should We Trust Capstone. In Advances in Cryptology-Crypto '96, N. Koblitz (Ed.), LNCS 1109, pages 89–103, Springer-Verlag, 1996.

[18] 17) A. Young. Kleptography: Using Cryptography Against Cryptography. PhD Thesis, Columbia University, 2002.