ShortPK: A Short-Term Public Key Scheme for Broadcast Authentication in Sensor Networks

Ronghua Wang, Wenliang Du, Xiaogang Liu
Syracuse University
and
Peng Ning
North Carolina State University

Broadcast authentication is an important functionality in sensor networks. Energy constraints on sensor nodes and the real-time nature of the broadcasts render many of the existing solutions impractical: previous works focusing primarily on symmetric key schemes have difficulties in achieving real-time authentication. Public Key Cryptography (PKC), however, can satisfy the real-time requirements, and recent trends indicate that public key is becoming feasible for sensor networks.

However, PKC operations are still expensive computations. It is impractical to use PKC in the conventional ways for broadcast authentication in sensor networks. To reduce costs, we propose *ShortPK*, an efficient Short-term Public Key broadcast authentication scheme. The basic idea is to use short-length public/private keys, but limit their lifetime to only a short period of time. To cover a long period of time, we need to use many public/private key pairs; distributing these public keys to sensors is a challenging problem. We describe a progressive key distribution scheme that is secure, efficient, and packet-loss resilient. We compare our scheme with the traditional 160-bit ECC public-key schemes, and show that our scheme can achieve a significant improvement on energy consumption.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms: Security, Design, Performance

Additional Key Words and Phrases: Sensor Networks, Public Key, Communication

1. INTRODUCTION

Wireless sensor networks are being used in a wide variety of applications, such as military sensing and tracking, environment monitoring, patient mornitoring, etc. Usually, sensor networks are composed of one or more base stations and a number of sensor nodes. The base stations serve as the commanders and the data sinks, which broadcast commands to

Authors's addresses: Ronghua Wang, Wenliang Du, and Xiaogang Liu, Department of Electrical Engineering and Computer Science, 2-120 Center for Science and Technology, Syracuse University, Syracuse, NY 13244; email: {rwang01, wedu, xliu08}@ecs.syr.edu; Peng Ning, Department of Computer Sciense, 3320 Engineering Building II, North Carolina State University, Raleigh, NC 27695; email: pning@cs.ncsu.edu.

sensors; the sensor nodes, upon receiving those commands, will send the results back to the base stations. When deployed in hostile environments, sensor networks are susceptible to a variety of attacks. For example, adversaries can easily listen to the communication between the sensor nodes and the base stations, impersonate base stations to deceive the sensor nodes, or send back wrong information to fool the base stations. Many countermeasures have been developed to defeat or remedy these attacks; among them, broadcast authentication is intended to prevent attackers from impersonating base stations. The goal is to authenticate the broadcast messages, and make sure they are indeed sent by the base stations (base stations are usually secured).

Broadcast authentication originated from the applications in the Internet environment, but energy constraints on sensor nodes and the real-time nature of the broadcasts in sensor networks render many of the existing solutions impractical. A number of broadcast authentication schemes have been proposed [Gennaro and Rohatgi, Wong and Lam, Miner and Staddon, Song et al., Golle and Modadugu, Canetti et al., Perrig et al., Rohatgi], some for the Internet and some for sensor networks. However, these schemes are not the perfect solutions for broadcast authentication in sensor networks due to the resource constraints of sensor networks.

An ideal broadcast authentication scheme for sensor networks should satisfy the following requirements:

— *Computation efficiency*: sensor nodes usually do not have powerful CPUs or sufficient power to conduct expensive computations. Even if they do, the amount of energy consumed is quite significant.

— *Communication efficiency*: broadcast in sensor networks is usually conducted in a relay fashion because sensor nodes may not hear the base station directly. If broadcast authentication turns a packet into a much larger one, energy cost on communication becomes very expensive.

—*Packet-loss resilience*: packet loss is more prominent in sensor networks than in the Internet environment; it can be caused by unreliable wireless communication, or even worse, by malicious jamming. Therefore, being able to tolerate packet loss is extremely important.

—*Real-time*: broadcasts in sensor networks are usually real-time, so for senders, once a message is ready, it should be transmitted without delay; for receivers, once a message is received, it should be authenticated immediately.

—*Intrusion resilience*: broadcasts in sensor networks are especially susceptible to intrusions; attackers can broadcast any message, meaningless or not, to the sensor nodes. So intrusion resilience should be an important property of a broadcast authentication scheme.

Most existing broadcast authentication schemes do not satisfy the aforementioned properties. For example, Perrig et al. proposed μTESLA in [Perrig et al. 2001], which is quite efficient and robust to packet loss, but it cannot achieve real-time authentication. Public key cryptography (PKC), on the other hand, is desirable for broadcast authentication. However, public key operations had been widely considered as impractical for sensor networks because of their high computation costs and large signature size. Recent research has shed light on the public key schemes in sensor networks [Gura et al., Wander et al., Ning]. For example, it was pointed out that Elliptic Curve Cryptography (ECC) is computationally

feasible for MICA2, a popular CPU used in Berkeley motes [Wander et al.]. With the advance of hardware technology, it is reasonable to assume that future sensor nodes will be equipped with PKC implementations.

However, as practical as they may become, PKC operations are still expensive for sensor nodes; the cost of public key operations may dominate the cost of transmitting packets in sensor networks. Since broadcast authentication requires sensors to conduct a large number of PKC operations, using PKC in the traditional ways is still impractical. It is desirable if we can significantly reduce the cost of PKC operations. There are two viable approaches toward this goal: one is to optimize the implementation of public key algorithms, like what has been done in existing works [Wander et al., Gura et al.]; alternatively, we can optimize the broadcast authentication protocols. In this paper, we take the latter approach.

We propose *ShortPK*, an efficient Short-term Public Key broadcast authentication scheme. Our basic idea is to use short-length public/private keys. This will reduce the security strength of public keys; so, instead of the traditional approach that uses one long key that is hard to break for a long period of time measured by years, we limit the lifetime of short public keys to a small period of time that is measured by minutes. We call the lifetime of each short public key a *term*. We divide the lifetime of a sensor network into a number of such terms, and we assign a different public/private key pair for each term. A public key is only valid within its own lifetime, so broadcast authentication is still safe even if a public/private key pair is broken after its term, because any signature generated using that pair will be invalid.

With the ShortPK approach, broadcast authentication becomes less expensive, but we face the challenge of distributing these public keys to sensors. The reason is, due to memory limitation, we cannot preload all the public keys into sensor's memory prior to deployment; even if we can, they must be kept secret until their corresponding terms. Therefore, the public keys need to be broadcasted by the base stations. *The broadcasted public keys must also be authenticated*. This becomes another broadcast authentication problem. However, there is an essential difference between this new broadcast authentication problem and the original broadcast authentication problem: this is an *offline* broadcast authentication problem. In this new problem, the broadcast messages are the public keys; *all these public keys can be generated offline*, even before sensors are deployed. In the original broadcast authentication problem, however, the broadcast messages are generated *online* (i.e. in real-time); it is unrealistic to assume that they can be generated offline. Solving offline broadcast authentication problems can be easier than solving online broadcast authentication problems. Therefore, in the ShortPK approach, the original message broadcast authentication problem is reduced to public key distribution problem. We propose a progressive public key distribution scheme (PPKD) in this paper. Our scheme is secure, efficient, and packet loss resilient.

Organization The organization of our paper is as follows: Section 2 discusses the related work, and in Section 3 we present the basic idea of the ShortPK scheme. We describe a progressive key distribution scheme in Section 4. Section 5 presents the evaluation results of our scheme, and finally, Section 6 concludes the paper.

2. RELATED WORK

A number of broadcast authentication schemes have been proposed, especially for the Internet environment. Gennaro et al. proposed a one-time signature scheme for stream

authentication in [Gennaro and Rohatgi]. In this scheme, packets are considered as a sequence, packet M_i carries a one-time public key for packet M_{i+1}, after the receiver verifies M_i, it buffers the one time public key for M_{i+1}. The proposed one-time public key scheme is efficient in signature signing and verification, but when used in sensor networks, it has two disadvantages. First, the scheme does not tolerate packet loss; if one packet is lost, all the future packets cannot be verified. Second, signatures generated by the proposed one-time public key scheme are quite large; they are more than 1000 bytes long.

One approach to reduce the cost of public-key based signature signing and verification is to sign a number of packets together. In this way, the amortized cost of signature is reduced. A number of amortized approaches have been proposed, including [Wong and Lam, Miner and Staddon, Song et al., Golle and Modadugu]. A common property of the amortized approach is the need for buffering (either sender buffering or receiver buffering). In many applications, such as the authentication of digital streams, the sender knows the whole packet sequence, buffering is reasonable. However, in sensor networks, broadcast messages are usually unpredictable and time sensitive. In these cases, sender and receiver buffering will compromise the performance of the broadcasting authentication. The scheme we present in this paper does not require buffering; signature signing and verification can be conducted instantly.

Message authentication codes (MAC) has been proposed as another approach. In a MAC approach, senders and receivers share a secret key; senders append a MAC to each message. This approach is not secure in sensor networks, because if one sensor is compromised, the master key will be compromised. An improved MAC scheme is described in [Canetti et al. 1999]. Instead of using one MAC key, a key pool is suggested in this scheme to improve the robustness against spoofing. However, the communication overhead in this scheme is undesirable because the number of MAC keys attached to each packet is as large as the key pool size.

To achieve broadcast authentication which is both efficient and resilient to packet loss, Perrig proposed BiBa [Perrig 2001], which provides very fast signature verification. It exploits the birthday paradox to achieve efficiency and security. Signature verification of BiBa is very efficient, but the size of public keys in BiBa is very large, and so is the communication overhead. In contrast, our scheme has much shorter public key sizes and much smaller signature size.

Perrig et al. [2001] proposed μTESLA for broadcast authentication in sensor networks in. Based on one-way hash chain of commitments, μTESLA is resilient to packet loss and has a low communication overhead, but receivers cannot verify signature instantly. In [Perrig et al. 2001], μTESLA was extended to an immediate authentication mechanism by replacing receiver buffering with sender buffering. As we discussed earlier, sender buffering is not suitable for broadcast authentication in sensor networks. Compared to the hash-chain based μTESLA scheme, our scheme is less efficient; however, broadcast authentication in our scheme can be instant without buffering in either sender or receiver side.

Previous schemes on multicast authentication are also shown in [Canetti et al., Rohatgi, Wong and Lam]. However, these schemes are mostly impractical for sensor networks due to their packet-loss tolerance, computation costs, and communication costs.

3. BROADCAST AUTHENTICATION USING SHORT-TERM PUBLIC KEYS

To make a public key system secure, keys need to be long enough. Most of the public key systems in use today use 1024-bit parameters for RSA, while 160-bit ECC keys can achieve the same security level [Gura et al.]. The costs of using the traditional public key scheme in sensor networks are high in both computation and communication. The high costs are due to the length of the public/private keys, which not only makes computation complicated, but also makes the length of signatures long, and thus requires more energy to transmit the signatures. [1]

To reduce the costs while still using the existing public key algorithms for sensor networks, we can reduce the length of public/private keys. However, doing so compromises security. Traditional public key schemes make sure that the public/private keys are long enough, so it takes very long time to breaking the scheme (i.e., finding the private keys, forging signatures, or decrypting ciphertexts). In these schemes, "long time" means years. For encryption, "long time" is necessary; if the encrypted information should be kept secret for n years, then the public keys cannot be broken within n years.

However, if we only use public keys for signatures, rather than for encryption, we do not need the public key to be strong for such a long time. This is because the signature itself (the product of signing) is not confidential; it is the signing process that we need to protect. In other words, we only need to guarantee that the public key is hard to break between the signing time and the signature delivering time. If we can put a small bound on this time window, we can use public/private keys with a much shorter length.

We propose a short-term public key scheme for broadcast authentication for sensor networks. We divide the lifetime of a sensor network into short terms, and we use one public key for each term. We also refer to a term as the lifetime of a public key. Public keys in different terms are independent and will only be disclosed to the receivers during their corresponding terms. A public key must be unbreakable during its lifetime (plus the bound of communication delay and the bound of error in clock synchronization). Because the lifetime requirement for a public key system is reduced to hours or even minutes instead of years, we can significantly reduce the length of public/private keys, and thus lower the costs, especially the computation cost. For example, if we reduce an ECC public key from 160 bits to 80 bits, the computation cost for signature verification is reduced to roughly one eighth and the length of signatures is reduced to half. As a tradeoff, such an ECC public key may only survive tens of minutes. By limiting their life to a short period of time, the signature is still reasonably secure.

3.1 Security Strength of Short Public Keys

Shorter public keys reduce the security strength of their public key systems. In this subsection, we study the relationship between key length and security strength. Such relationship can serve as the guideline for us to choose key length and the lifetime for each public key.

3.1.1 *Short exponent RSA.*

In certain applications where there is a large difference in computing power between two communication devices, the idea of using short public components or secret components is natural. The typical example is the RSA used in communications between a smart card

[1] Although in RSA, the public key e can be very small, which makes the signature verification much more efficient than signature generation, but the length of signature is too large compared to ECC.

and a large computer [Wiener], where it would be desirable for the smart card to have a short secret component and the computer to have a short public exponent in order to reduce the processing for smart card. However, there is a risk in this practice: short RSA keys are vulnerable to attacks.

There have been some researches focusing on the security weakness of the short exponent RSA [Blomer and May, Boneh and Durfee, Wiener]. For example, Weiner pointed out in [Wiener] that if the private exponent d used in RSA is less than $N^{0.25}$, then the system is insecure. Boneh et al. further studied the problem and proved that if the private exponent d is less than $N^{0.292}$ then the system is not secure [Boneh and Durfee]. Further more, in their paper, Boneh believed that using their approach, security system is not secure as long as $d < N^{0.5}$.

Compared to the study of short exponent RSA, as far as we know, other than the fact that the security strength of the public keys will deteriorate with the reduction of key sizes, there is no active study in the field of short ECC keys yet.

3.1.2 *Short ECC keys.*

In our studies, we focus on how long it takes an adversary to find the private key from a public key in ECC. This problem is equivalent to break the elliptic curve discrete logarithm problem (ECDLP). It is widely believed that the ECDLP problem is computationally hard to solve when the point used in the elliptic curve has large prime order. Among all the know algorithms to break ECDLP, such as Shanks' baby-step-giant-step method [Shank], Pollard's methods [J.M.Pollard], the Menezes-Okamoto-Vanstone (MOV) attack [Menezes et al.], the Pohlig-Hellman algorithm [Washington], the only algorithms that are applicable for all elliptic curves are the methods of Shanks and Pollard, and these methods have exponential complexity.

General purpose processors can be used to execute ECC operations; for example, Pentium 100MHz machines can perform roughly 16000 operations of 89-bit elliptic curves [Certicom]. However, several processors have been designed specifically for the ECC operations, such as [Wolkerstorfer, Eberle et al., Puhringer]. One of the fastest ECC-processors is presented by Eberle et al. [Eberle et al.], which is a very powerful processor that can also perform RSA calculations: using a 64 bit multiplier, it achieves a very high performance of 6000 ECC-operations per second for public keys of 224-bit long. The implementation was based on the then-current processor technology of 1.5 GHz. Notice that the sizes of the ECC keys are different in the above examples: the longer the size of public keys, the smaller number of ECC-operations the processor can execute within the same period of time.

Let T_m denote the time for an attacker to break an m-bit ECC challenge problem. That is, T_m is the time to find out the private key from a given public key. An improved Pollard's rho algorithm takes approximately $\sqrt{\pi 2^m}/2$ elliptic curve addition operations to solve a key challenge problem with m-bit key length. Assume the ECC-processor presented in [Eberle et al.] is used to break the elliptic curve key, we can estimate T_m (in *machine days*) as the following:

$$T_m = \frac{1}{6000 \times 60 \times 60 \times 24} \times \frac{\sqrt{2^m \pi}}{2 \times (\frac{\lceil \frac{224}{32} \rceil}{\lceil \frac{m}{32} \rceil})^2} \approx 10^{-5} \times \frac{2^{\frac{m}{2}}}{6000 \times (\frac{7}{\lceil \frac{m}{32} \rceil}^2)} \quad . \qquad (1)$$

In the above equation, $(\frac{7}{\lceil m/32 \rceil})^2$ is an adjustment for the m-bit elliptic curves; this

Table I. Time to break an m-bit ECC public key using ECC-processor in [Eberle et al.] (in $MachineDays$)

Key Length ($bits$)	64	80	89	96	112	128	160
T_m(*machine days*)	0.5	233	4360	86166	3.01×10^7	1.01×10^{10}	1.02×10^{15}

adjustment is based on the fact that on a 32-bit machine an elliptic curve addition in an m-bit field takes $(\frac{\lceil 224/32 \rceil}{\lceil m/32 \rceil})^2$ as much time as that in an m-bit field. Table I shows the estimated number of machine days to solve an m-bit ECC challenge problem based on the above equation.

Using the data from Table I, we can choose a suitable size for our short public keys. For instance, based on the table, it will take 233 machines to work together for about 24 hours to break an 80-bit ECC key. Therefore, if the lifetime of an 80-bit key is set to 5 minutes, to break it within this lifetime, adversaries need at least 66,000 computers to work simultaneously. If this kind of risk is still unacceptable to an application, or if the adversary use more powerful machines, we can further shorten the lifetime.

3.2 The Key Distribution Problem

With short-term public keys, a sensor network needs a lot of public keys. Distributing these keys securely to sensors is a challenging problem. Using the traditional public key schemes, there is only one public key, which can be preloaded into sensors' memory before they are deployed. In our scheme, we have a number of public keys, sensors might not have enough memory to store all of them. Even if they do, we cannot simply preload those public keys in the same way as we do in the traditional schemes, because short public keys need to be kept secret before their terms; otherwise, attackers can get the public keys by compromising a sensor, and might break these public keys before the keys' lifetime periods. Therefore, a secure and efficient key distribution scheme is needed for the short-term public key scheme.

Naively, we can let the base stations simply broadcast the public keys to sensors during their corresponding terms. However, this is not secure because of the well-known *man-in-the-middle* attack: without any means to authenticate the public keys, sensors cannot tell whether the received public keys belong to the base stations or not, because attackers can impersonate the base stations and send their own public keys to sensors. In the Internet environments, public keys are authenticated using certificates. We cannot use the certificates; otherwise, our scheme will be more expensive than the traditional public-key-based broadcast authentication schemes. We need a more efficient solution to distribute the short-term public keys. We formulate our problem in the following:

Problem 1 *(The Public Key Distribution Problem) The base station has N public keys Pk_i, for $i = 1, \ldots, N$. These public keys need to be distributed to the sensor network. Sensors need to authenticate the public keys efficiently.*

This problem becomes another broadcast authentication problem. As we discussed earlier, there is an essential difference between this problem and the original broadcast authentication problem that we are trying to solve in this paper. This problem is an *offline* broadcast authentication problem, i.e., the broadcast messages (the public keys) are known *a priori*, and can be generated offline. In the original broadcast authentication problem, messages are usually unknown *a priori*.

A straightforward solution to this problem is to use Merkle tree [Merkle] which turns the public-key-based certificate scheme into a much more efficient hash-function-based scheme. However, the communication overhead is undesirable for sensor networks because the size of the certificate for each public key is large: $O(\log N)$, where N is the number of public keys in the Merkle tree. Although this does not have much effect on senders because senders (base stations) usually have sufficient power, it does increase the power consumption on sensors: sensors not only need to listen to this larger message, but also need to relay it for those who cannot hear the base station directly. The energy overhead caused by listening and relaying the certificates might negate the savings on the computation. Therefore, from the energy point of view, Merkle tree is not a practical solution to our public key distribution problem.

One way to save energy in the Merkle tree solution is to let sensor nodes store the public keys that they have received (and authenticated). These keys will be served as the proof of the future public keys; to verify an incoming public key, the communication cost is minimized: only one additional key is transmitted. One of the most important shortcomings of this approach is that this approach is not resistant to packet loss: if one public key is lost in the transmission, we will not be able to authenticate future public keys. Another issue of this solution is the memory cost: it could be very large because the sensor nodes may need to store all the nodes in the Merkle tree. Du, Wang and Ning studied this issue in [Du et al.]. So from the point of view in packet-loss resiliency and the memory requirement, this approach is not quite feasible in sensor networks.

Another solution is to use the Graph-Based Scheme (GBS) [Miner and Staddon, Song et al.]. GBS considers packets as vertices in an authentication graph. In this graph, a directed edge $\overrightarrow{e}(i,j)$ denotes that packet M_i carries a hash of packet M_j, thus M_j can be verified by the hash if M_i is received and verified. One of those packets, denoted by M_{sig}, carries a signature generated by a public key algorithm. M_j can be verified if there is at least one path from M_{sig} to M_j.

One of the important properties GBS needs to maintain is the resilience to packet loss. In the Internet environment, packet loss is mostly due to unreliable broadcasting, but in sensor networks, packet loss can also be caused by malicious attackers; even worse, attackers can launch selected jamming attacks that only target at a few selected ("important") broadcasting packets. While this type of attack might not be viable for the Internet environment, it is an effective attack in sensor networks. If there are important broadcasting packets, whose loss can have catastrophic impact, attackers can selectively target those packets. The GBS schemes are subject to this type of selective attack because in GBS, the packets broadcasted at the early stage are more important than those at the later stage. In particular, the first few packets (e.g. M_{sig}) are extremely important; if a sensor does not receive these important packets, it will be unable to authenticate future broadcast messages. Therefore, in our scheme, in addition to tolerate random packet loss, we also need to tolerate targeted packet loss. We can achieve this goal by exploiting a unique property of sensor networks that does not exist in the Internet environment: that is, sensors usually belong to the same authority before their deployment; thus, a certain degree of trust can be bootstrapped before sensors are deployed in the field.

To recover the lost public keys broadcasted during a specific term due to packet loss, one solution is to let the sensor nodes query their neighbors once they find out they do not receive a message. This approach has been adopted in many researches, such as [Intanagonwiwat et al.]. However, some reasons prevent this approach from being effective:

(1) Communication overhead associated with this approach: now sensor nodes need to query their neighbors each time they think they miss a message, and this will be a huge burden for the sensor nodes; (2) The delay associated with the query: query messages involve sending out and receiving query messages, in addition to the processing time of the sensor nodes. This will incur extra delay for the sensor nodes; (3) The security issue related with this approach: malicious nodes may send out faked public keys when they receive a query message, so the honest nodes are very likely to be fooled. For the above reasons, we do not adopt the the query-neighbor approach in our study.

4. THE PROGRESSIVE PUBLIC KEY DISTRIBUTION SCHEME (PPKD)

4.1 The Basic Scheme

Assume N public keys will be used during the lifetime of a sensor network. We use PK_i, for $i = 1, \ldots, N$ to denote these keys. Each public key has a fixed lifetime. We use $t_1, \ldots, t_N$ to represent the starting time of each public key's lifetime, i.e., PK_i's lifetime starts at t_i and ends at t_{i+1}. Because of delay in broadcasting and errors in clock synchronization (between sensors and base stations), the actual lifetime for PK_i should take these into consideration. Assume that synchronization errors are bounded by σ_s, and broadcasting delay is bounded by σ_b, the lifetime of PK_i is indeed $[t_i - \sigma_s, t_{i+1} + \sigma_s + \sigma_b]$. Since usually $t_{i+1} - t_i$ is relatively large compared to σ_s and σ_b, the estimation of these bounds does not need to be tight, and the clock synchronization does not need to be very accurate. For the sake of simplicity, we omit these bounds in the rest of the paper, and refer lifetime periods of a public key using just t_i's.

Let us first assume that sensors have enough memory to hold all these N short-term public keys. This is an impractical assumption in many applications, and we will lift this assumption in our next scheme; this scheme only serves as a base for our next scheme. With enough memory, we load all of the N public keys into sensor's memories before sensor deployment, but the public keys must be encrypted, such that nobody, including all the sensors, knows PK_i before time t_i, the starting time of its lifetime. The encryption keys, denoted as $K_1, \ldots, K_N$, are symmetric encryption keys. If the public keys are not encrypted, adversaries can immediately get all the public keys, and will have much longer time to find the corresponding private keys. Encryption prevents adversaries from obtaining the public keys that have not yet become "alive".

When the time reaches t_i, the key PK_i becomes alive, so the base station needs to disclose the encryption key K_i to sensors. This can be achieved by attaching K_i in each of the broadcasting packets during $[t_i, t_{i+1}]$. *However, K_i must also be verified by sensors; otherwise, the broadcasting is not secure at all using the following attacks*:

(1) Before t_i, the adversary obtains $(PK_i)_{K_i}$, the encrypted public key PK_i. This can be done by capturing a sensor.

(2) The adversary randomly chooses a key K', decrypt $(PK_i)_{K_i}$ using K', and gets PK'. Obviously, PK' is gibberish because K' is different from K_i. However, the adversary treats PK' as a public key and finds its private key using the brute force method. If the length of private key is short, the adversary might be able to find the private key before t_i.

(3) At time t_i, the adversary broadcasts a malicious message, signed by the private key derived from PK'. The key K' is also attached to the message.

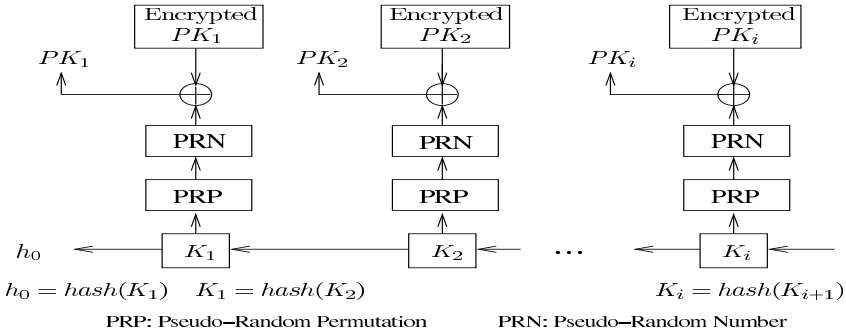


Fig. 1. Decrypting Public keys using verified symmetric keys.

(4) If sensors do not verify K', they will get the same gibberish PK' if they decrypt $(PK_i)_{K_i}$ using K'. They will use PK' as the public key to conduct signature verification. The verification will be successful, so sensors will accept the malicious message.

Therefore, sensors need to verify whether the received K_i is from the base station or not. This can be achieved by using one-way hash chain. Let K_i's be the nodes on a one-way hash chain, with $K_i = hash(K_{i+1})$. The end (denoted as h_0) of the hash chain is pre-loaded into sensor's memories. When receiving a secret key K_i', the sensor just need to verify whether $hash^i(K_i')$ is equal to h_0 or not [2]. If not, the received K_i' is not the authentic K_i, and must be discarded. Figure 1 depicts the one-way hash chain.

We can use AES for the symmetric-key encryption, but alternatively, we can use exclusive OR (XOR) because each K_i is a one-time key (public keys are encrypted using different secret keys). It is possible that the length of K_i is not enough to encrypt a public key. For example, if we use an 80-bit hash function, the size of K_i will be 80 bits, which is not enough to encrypt a 100-bit public keys. This problem can be solved by using K_i as a seed to generate a pseudo-random bit sequence of the required length, and use this bit sequence as the key to encrypt and decrypt the public keys. Many papers have proposed different ways to generate the pseudo-random bit sequences, such as [Hall et al., Naor and Reingold, Luby and Rackoff]. We use the approach proposed in [Hall et al.] to generate the required length pseudo-random bit sequences; that is, we first use K_i as a seed to generate a hash value, and then process the hash function output with a pseudo-random permutation (PRP). After that, we use the pseudo-random number (PRN) generator to generate the bit sequence of the required length. We depict the process in Figure 1.

In the above example, we use a weak 80-bit hash function to create a one-way hash chain. Recent researches have shown that it is not very difficult to find multiple data with same hash value [Wang and Yu, Wang et al., Wang et al.]. We notice that these researches only break the collision-free property of hash functions, but so far, the one-way property remains secure: it has not been broken yet. Since in our scheme, we mainly utilize the one-way property of the hash chain instead of the collision-free property, we reasonably assume that the 80-bit hash functions used in our scheme is secure enough to achieve the desired security level necessary.

[2] $hash^i(x)$ is defined as $hash^{i-1}(hash(x))$. It is well known that we do not need to conduct i hash functions: since $hash(K_i) = K_{i-1}$, if K_{i-1} has already been verified, K_i can be verified using K_{i-1} with just one hash.

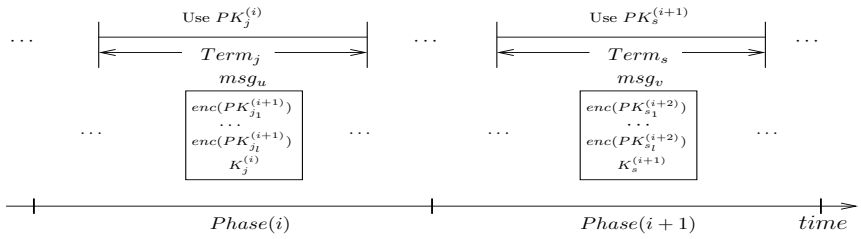


Fig. 2. Illustration of the progressive scheme

As we said before, this basic scheme assumes that sensors have enough memory to hold all the encrypted public keys. Although this assumption is impractical in general, it is feasible for some applications in which sensor nodes are only useful for a short period of time. For example, in typical rescue missions, a sensor network is quickly deployed to help the rescue. The lifetime of sensor nodes is expected to be within several hours to one day. If each public key's lifetime is 10 minutes, then 150 public keys will be sufficient for the entire mission (the total memory usage for 80-bit ECC public keys is 3K bytes). Assuming one message will be broadcasted per minute, these 150 public keys will be able to authenticate 1,500 messages. In this case, it is feasible to pre-load all the public keys (in encrypted forms) into sensor's memories. Therefore, apart from serving as the base for our next improved scheme, this scheme has its own merits in these special situations.

4.2 A Progressive Public Key Distribution Scheme (PPKD)

When the number of public keys is large, sensors do not have enough memory to hold all the public keys. Therefore, some public keys have to be distributed by the base station after the deployment. If there is no packet loss during broadcasting, the simple chaining approach is sufficient: we broadcast PK_i's hash before its lifetime t_i, along with a signature produced using PK_{i-1}'s private key. Sensors can authenticate the hash using PK_{i-1}, and later authenticate PK_i using the hash. This works perfectly if sensors can receive every broadcasting packet. However, if a sensor fails to receive one public key, it will be unable to authenticate any future public key, and hence unable to authenticate future broadcasting messages either. We propose a progressive scheme to achieve efficient public key distribution in a lossy communication environment.

We divide the entire lifetime of a sensor network into time periods with a fixed length; we call them *phases*, and use Ph_i to represent the i-th phase. Each phase is further divided into n time slots, and we call them *terms*. We use T to represent a term, $PK_j^{(i)}$ to refer to the public key used in the j-th term (denoted as T_j) of phase Ph_i, and $K_j^{(i)}$ to refer to the symmetric key used in T_j of phase Ph_i. A public key is only "alive" in its own term, so we also call a term the lifetime of a public key. During each term, a number of messages might be broadcasted by the base station, and we use $|T|$ to represent the number of messages broadcasted during each term. The relationship between *phase* and *term* is depicted in Figure 2. Our main idea for key distribution is to broadcast the encrypted public keys for phase Ph_{i+1} during phase Ph_i. The protocol is described in the following:

1. Setup Prior to deployment, each sensor is preloaded with the encrypted public keys of the initial phase Ph_0. There are n public keys for each phase.

2. Authentication At term T_j during phase Ph_i, the public key $PK_j^{(i)}$ should be disclosed

to each sensor. This is achieved using the basic scheme described in Section 4.1. Namely, we attach the symmetric key $K_j^{(i)}$ with each broadcast packet; sensors use the hash chain to authenticate the symmetric key, and then use this symmetric key to decrypt the encrypted public key $PK_j^{(i)}$ that is already stored in their memories. Finally sensors use $PK_j^{(i)}$ to authenticate broadcast messages within the term T_j.

3. Key Distribution In addition, during T_j, the base stations also broadcast a set of public keys (encrypted) for the next phase (Ph_{i+1}). These keys are carried along with the broadcast messages, and signatures of the packets also include these public keys. As long as the receiving sensors have the public key $PK_j^{(i)}$, they can verify the signature to make sure that both message and the encrypted public keys are authentic. Figure 2 also depicts this process.

If the lifetime of sensor nodes is expected to be one week, and if each public key's lifetime is 10 minutes, then the total number of public keys needed is about 1000. Assuming one message will be broadcasted per minute, these 1,000 public keys will be able to authenticate 10,000 messages.

To resist packet loss, keys must be sent with a certain degree of redundancy. We can use sophisticated error correcting codes to minimize communication overhead; however, the increased decoding cost might negate the intended savings of computation cost. For example, Havinga studied two types of error correction code, EVENODD and Reed-Solomon, and pointed out that compared with the computation cost, communication cost is negligible [Havinga]. It is estimated that the energy cost to transmit 1-bit packet using Intel StrongARM 1100 processor is about $1\mu J$ [Yu and Pransanna], which means transmitting a packet 10-bytes long about $80\mu J$, while the energy cost of implementing error correction using either EVENODD or Reed-Solomon codes may take up to several hundred mJ. We can see that the difference between the error correction code and transmission cost is several magnitudes. Therefore, we decide to use a straightforward but much more computation-efficient scheme, i.e., simply broadcasting each public key multiple times: within each term T, we let those $|T|$ broadcast messages together carry a total of k different encrypted public keys. On average, each public key is sent $k * n/n = k$ times (n is the number of terms in each phase, and is also the number of public keys in each phase, because one public key is used for each term). The cost of sending these k public keys is amortized by the $|T|$ messages; on average, each message carries $\frac{k}{|T|}$ public keys. The packet format of the ShortPK scheme is depicted in Figure 3. In this figure, L_{PK} is the size of public key and $L_{decrypt}$ is the size of symmetric key.

It should be noted that since $|T|$ is usually larger than k, each packet in practice carries either one or zero public key. Therefore, the length of each packet is either $4L_{pk} + L_{decrypt} + 12$ bits or $2L_{pk} + L_{decrypt} + 4$ bits. To further reduce the size of a packet, we do not need to carry the decryption key $K_j^{(i)}$ for each packet, because the key is the same for all the packets during term T_j. Due to potential packet loss, we need to let more than one packet carry this decryption key. If in each term we let z of $|T|$ packets carry the decryption key, the average packet size can be further reduced by $(1 - \frac{z}{|T|})L_{decrypt}$, while the chance that all of them get lost is q^z if packet loss is uniformly random and the packet loss rate is q. For the sake of simplicity, we do not consider this improvement in our evaluation.

We want to emphasize that in the proposed solution, it is possible that packet loss still exist. Resistant to packet loss is one of the design goals of our scheme; in hostile environ-

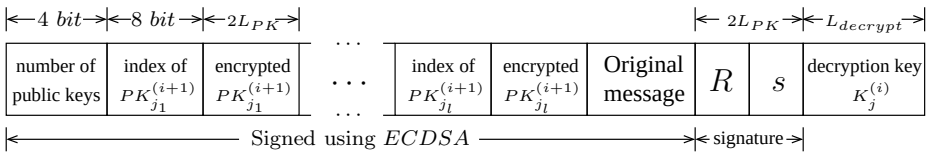


Fig. 3. Format of broadcast packets in the term T_j of phase Ph_i: (R, s) is the $ECDSA$ signature.

ment where packet loss is inevitable, to achieve packet-loss resistant, we always need to make tradeoff between the packet loss rate and the cost associated with the packet loss rate. Our scheme is addressing this issue with redundancy by sending same key multiple times. We can adjust the parameters of the scheme so that the impact of packet loss is reduced to the minimum (based on the cost allowed).

4.3 Selecting Public Keys in Each Phase

In the PPKD scheme, in each term T, k different encrypted public keys for the next phase are broadcasted along with the broadcast messages. The choice of these k public keys is important to the packet-loss resilience property of the next phase. We study the following two approaches and analyze their packet-loss resilience: (1) selecting k public keys randomly; (2) selecting k keys randomly, but guaranteeing that all the public keys are selected *equal number of times*. In each phase of the PPKD scheme, there are n public keys to send during the n terms in each phase. If we send k public keys ($k < n$) in each term, the average number of times each public key is sent is k. The second approach guarantees that each public key is sent *exactly* k times. We call the first approach the ***Random-K*** scheme and the second one the ***Exact-K*** scheme.

In the PPKD scheme, if broadcasting is lossless, the public keys for phase Ph_i will all be in sensors' memory after phase Ph_{i-1}. However, because packets can be lost, after phase Ph_{i-1}, it is likely that some public keys (say PK) for phase Ph_i will not be in a sensor's memory. This is caused by two reasons: (1) all the packets carrying PK are lost; (2) one or more packets carrying PK are received, but the sensor cannot verify the signatures of these packets. We use P_i to denote the probability that a public key PK of phase Ph_i is in memory after phase Ph_{i-1}. Since the loss of PK prevents a sensor from verifying signatures during its corresponding term in phase Ph_i, we also call P_i the *verification ratio* for phase Ph_i. We can calculate P_i using the following theorems (we use q to represent the packet loss ratio):

THEOREM 4.1. *For the* Random-K *scheme, the verification ratio P_i for phase Ph_i is the following:*

$$P_i = 1 - \{(1 - P_{i-1}) + P_{i-1} \cdot [q + (1 - q)(1 - \frac{k}{n|T|})]^{|T|}\}^n.$$

The proof of this Theorem is long, so we put it in Appendix A.

THEOREM 4.2. *For the* Exact-K *scheme, the verification ratio P_i for phase Ph_i is the following:*

$$P_i = 1 - \{q + (1 - q) \cdot (1 - P_{i-1})\}^k.$$

PROOF. Since each public key PK (needed in Phase i) is guaranteed to be carried by exactly k independent packets during Phase $i - 1$, let us take any one of them (say M),

and compute the probability that M does not cause PK to be accepted into memory during Phase i. There are two events that prevent PK from being accepted: (1) the packet M is lost; (2) the packet M is not lost, but it cannot be verified because the sensor does not have the corresponding public key (say PK_M) in the memory. It should be noted that $\Pr$(M cannot be verified in Ph_{i-1}) is the same as $\Pr(PK_M$ is not in the memory during $Ph_{i-1})$, which is $1-P_{i-1}$, because the definition of P_{i-1} is $\Pr(PK_M$ is in the memory during $Ph_{i-1})$. Therefore:

$$\begin{aligned} & \Pr(\text{M does not cause PK to be accepted in } Ph_{i-1}) \\ = \ & \Pr(\text{M is lost}) + \Pr((\text{M is not lost}) \wedge (\text{M cannot be verified in } Ph_{i-1})) \\ = \ & q + (1-q) \cdot (1 - P_{i-1}). \end{aligned}$$

Since there are exactly k packets like M (i.e., they all carry PK), we have the following equation:

$$P_i = 1 - \Pr(\text{M does not cause PK to be accepted in } Ph_{i-1})^k = 1 - \{q + (1-q) \cdot (1 - P_{i-1})\}^k.$$

∎

Note that the above formulae of P_i are recursive because the right hands of the formulae contain P_{i-1}. It is difficult to obtain a closed form for P_i. However, the following theorems give us an important property of P_i:

LEMMA 4.3. *P_i is monotonic, for $i = 0, ..., \infty$.*

PROOF. To prove that P_i is monotonic, we just need to prove the following cases: (1) if $P_i > P_{i-1}$ then $P_{i+1} > P_i$ when $i > 0$; and (2) if $P_i < P_{i-1}$ then $P_{i+1} < P_i$ when $i > 0$.

By definition, we have $P_i = 1 - \{q + (1-q) \cdot (1 - P_{i-1})\}^k$ and $P_{i+1} = 1 - \{q + (1-q) \cdot (1 - P_i)\}^k$.

Therefore,

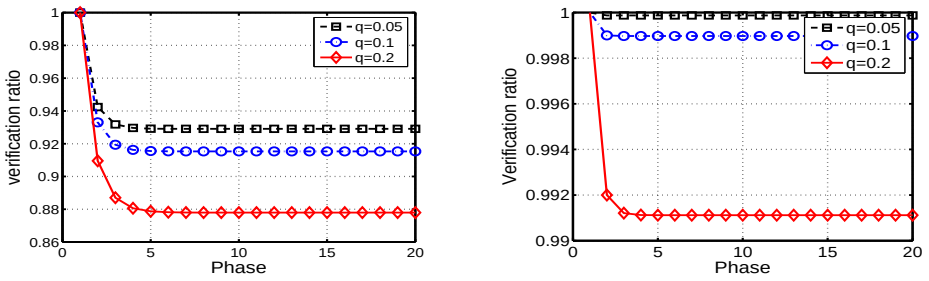
$$\begin{aligned} P_{i+1} - P_i \ &= \ \{1 - \{q + (1-q) \cdot (1 - P_i)\}^k\} - \{1 - \{q + (1-q) \cdot (1 - P_{i-1})\}^k\} \\ &= \ \{q + (1-q) \cdot (1 - P_{i-1})\}^k - \{q + (1-q) \cdot (1 - P_i)\}^k. \end{aligned}$$

We now prove case (1). That is, if $P_i > P_{i-1}$ then $P_{i+1} > P_i$ when $i > 0$:

$$\begin{aligned} \{P_i > P_{i-1}\} \wedge \{0 < q < 1\} \ &\Rightarrow \ \{(1 - P_{i-1}) > (1 - P_i)\} \wedge \{(1-q) > 0\} \\ &\Rightarrow \ (1-q) \cdot (1 - P_{i-1}) > (1-q) \cdot (1 - P_i) \\ &\Rightarrow \ \{q + (1-q) \cdot (1 - P_{i-1})\} > \{q + (1-q) \cdot (1 - P_i)\} \\ &\Rightarrow \ \{q + (1-q) \cdot (1 - P_{i-1})\}^k > \{q + (1-q) \cdot (1 - P_i)\}^k \\ &\Rightarrow \ \{q + (1-q) \cdot (1 - P_{i-1})\}^k - \{q + (1-q) \cdot (1 - P_i)\}^k > 0 \\ &\Rightarrow \ P_{i+1} - Pi > 0 \\ &\Rightarrow \ P_{i+1} > P_i. \end{aligned}$$

That is, we have proved case (1): if $P_i > P_{i-1}$ then $P_{i+1} > P_i$ when $i > 0$. Similarly, we can prove case (2): if $P_i < P_{i-1}$ then $P_{i+1} < P_i$ when $i > 0$. And from both cases, we know P_i is monotonic. ∎

Based on the fact that P_i is monotonic and bounded between 0 and 1, we have the following convergence theorem on P_i:



(a) P_i for Rondom-K scheme.

(b) P_i for Exact-K scheme.

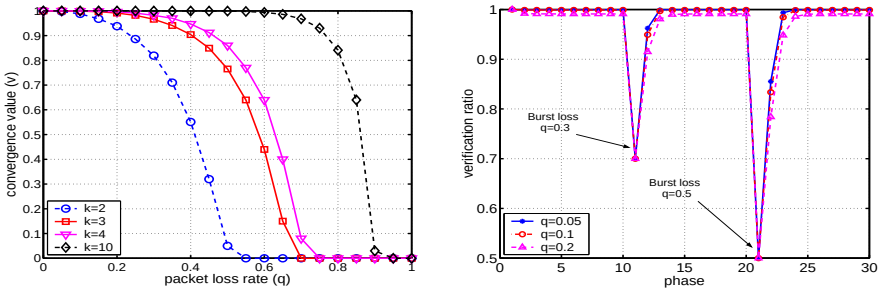
Fig. 4. Verification ration P_i for Random-K and Exact-K schemes ($k = 3$).

THEOREM 4.4. *P_i converges. If $P_0 = 0$, P_i converges to 0; if $P_0 \neq 0$, P_i converges to a number between 0 and 1.*

Figure 4(a) and 4(b) depict the results for the Random-K scheme and the Exact-K scheme, respectively, when the packet loss rate q changes from 0.05 to 0.20. The results clearly show that the Exact-K scheme is significantly better than the Random-K scheme. Therefore, in the rest of the paper, we assume that the Exact-K scheme is used. Figure 4(a) and 4(b) also show that P_i does converge. Although it is difficult to obtain a closed form for the convergence value for P_i, we can simply use numeric methods to find the values. The convergence values for the Exact-K scheme are depicted in Figure 5(a) for different k values and different packet loss rate q. The figure shows that the convergence value increases when k increases; the value decreases when the packet loss becomes more severe.

Packet Loss due to our scheme. Given the packet loss q, the actual packet loss q_{actual} is the sum of the packets that are lost during the transmissions and the packets that are discarded because a sensor cannot verify their signatures. Therefore, $q_{actual} = q + (1 - q)(1 - P_i)$. The first part is not caused by our scheme; only the second part is due to our scheme. We call the second part the additional packet loss. It was shown in Figure 5(a) that when k becomes larger and larger, $1 - P_i$ approaches zero exponentially. For example, when $q = 0.1$, if $k = 3$ public keys are sent within each term T, the additional packet loss caused by our scheme is only 0.1%. In other words, when a packet is received by a sensor, there is only 0.1% chance that the sensor cannot verify this packet.

The role of n. We also observe that in Exact-k scheme, the convergence value does not depend on n. It is interesting to know what role n plays in this scheme. Let us look at the phase i. The expected percentage of the public keys (for the next phase) received by each sensor is $P_i = P_i * n/n$. However, this is just an expected value, the actual occurrence of the events deviates from this value (thus we have variance); in other words, the actual percentage (denoted by P_{actual}) of the next-phase public keys received by each sensor might be lower than the expected value P_i. When P_{actual} becomes zero for some sensors during phase i, these sensors will never be able to authenticate any future broadcast message. The probability of this event is $(1 - P_i)^n$. Therefore, to make this event improbable, the value of n cannot be too small. On the other hand, $2n$ represents the total number of public keys that can be stored in a sensor's memory, so n is bounded by sensors' memory size.



(a) Convergence values for the Exact-K scheme.

(b) Tolerance against burst packet loss for the Exact-K scheme ($k = 3$).

Fig. 5. Convergence value and tolerance of burst packet loss for Exact-K schemes.

Tolerance of Burst Packet Loss. After we choose an appropriate value for n, although it is improbable for P_{actual} to be zero, it is still possible that P_{actual} becomes significant lower than the expected value P_i. If the packet loss is uniform random, the likelihood for P_{actual} to be small is rare (this likelihood follows the Binomial Distribution), but when the packet loss occurs in burst, it is highly likely that P_{actual} at Phase i deviates from P_i and becomes significantly lower than P_i. This will further affect P_{i+1}, P_{i+2}, and so on. However, regardless of what P_{actual} is at Phase i, the future P_{i+t} values will always (quickly) converge to the same convergence value so long as the burst loss rate is not 1. This is demonstrated in Figure 5(b), in which we emulate burst packet loss by intentionally letting the P_i's at certain points deviate from the theoretic values. The results show that the P_i values after the burst packet loss can quickly bounce back toward the original convergence value. This demonstrates the resilience of our scheme against burst packet loss.

It is possible that adversaries may implement jamming attack that targets at selected Phases: if all the keys in a specific Phase is lost, future public keys will not be able to be verified. Our scheme can handle the attack by adjusting the length of the Phases. We can increase the length of the Phases so that the adversaries cannot jam the communication of the whole Phases. So long as the packet loss rate during a Phase is not 1, our scheme can always recover from the packet losses. If the adversaries keep jamming the whole communication, then their cost will be dramatically increased, which might not be desirable for the adversaries.

5. PERFORMANCE EVALUATION

We compare the performance of ShortPK with the traditional public-key scheme. Since we assume that ECC is used, we use the performance of the 160-bit ECC scheme as our baseline. We focus on both computation cost and communication cost. As our scheme assume loose synchronization between the base station and the sensor nodes, synchronization will also consume energy. The added energy cost is mostly the added communication cost by the synchronization messages, which are independent of the broadcasted authentication messages. Research in the field of synchronization in sensor networks has produced many interesting and innovative approaches to implement efficient time synchronization schemes [Sichitiu and Veerarittiphan, Ostrovsky and Patt-Shamir, Dai and Han]. In real

applications, the synchronization will not be performed frequently, and these costs are independent from the broadcast messages, so in analyzing the synchronization costs, we can consider the costs amortized by the broadcast messages. We will discuss these costs in more detail in this section.

There are a number of parameters that can affect the performance of our scheme, including the broadcasting protocols, the actual packet loss rate (q and k), the number of messages broadcasted during the lifetime of each public key (i.e., the value of $|T|$), and elliptic curve domain parameters. In this section, we study how these parameters affect the performance of our scheme.

In our evaluation, the following notations are used to assist our analysis and discussion of the scheme:

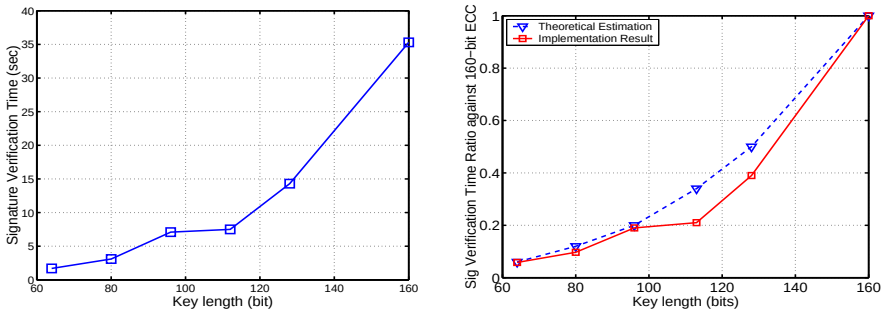
Notation	Explanation
N	Total number of one-time public keys
n	Number of short-term public keys in each phase
T	Lifetime of each individual short-term public key
$\|T\|$	Number of broadcast messages in term T.
q	Packet loss rate, $0 \leq q \leq 1$
k	Number of next-phase public keys sent in each term
P_i	Probability that a public key PK in phase i is in a sensor's memory
L_{pk}	Length of the public key
Ph_j	The jth phase
$[t_i, t_{i+1}]$	Lifetime of PK_i in a phase

5.1 Computation Cost

Computation cost is a critical issue in our scheme since our goal is to provide a broadcast authentication scheme that is both secure and efficient. We do not consider computation cost of base stations because we assume that the base stations are powerful machines without resource constraints. Sensor nodes' computation cost comes from two sources: (1) decrypting the public keys using symmetric-key cryptography, and (2) verifying the ECDSA signature. The time synchronization may also incur some computation cost, but the synchronization is not performed very frequently. Compared to public-key signature verification, the cost for symmetric-key decryption and the computation cost in synchronization is negligible. Therefore, we only consider the cost of public-key signature verification in our evaluation.

It is difficult to compute the energy consumption of the 160-bit elliptic curve operations directly. Noticing that the energy cost of the ECC operations is proportional to the number of instructions the CPU executes while the number of instructions the CPU executes is directly related to the time these instructions are executed, we take an alternative approach: we measure the energy consumption of the ECC operations by counting the time these operations execute. That is, in our evaluation, we use the processing time of the 160-bit ECC operations as the benchmark of the energy cost of the 160-bit ECC operations. We adopt two approaches to evaluate the signature verification cost: First, we present a theoretical estimation to compare the cost of ShortPK scheme with that of 160-bit ECC; and second, we implement the ECDSA algorithm on MICAz motes to measure the signature verification time for short keys.

An important property for ECC operations is that, computation cost is proportional to



(a) Signature verification time for elliptic curves with various key length.

(b) Signature verification time ratio against the 160-bit ECC curve.

Fig. 6. Computation cost (all the curves are generated using the CM method and the field primes p are chosen as pseudo-Mersenne primes).

the cube of key sizes [Vanstone]. Let us use $C(L_{pk})$ to represent the computation cost of public key operations with key size L_{pk}. Computation cost ratio of ShortPK scheme to the 160-bit ECC scheme can be estimated in the following:

$$\frac{C(L_{pk})}{C(160)} = (\frac{L_{pk}}{160})^3. \tag{2}$$

Equation (2) provides a theoretical guideline of how much computation advantage we can gain by using short keys. For example, if we use 80-bit ECC curves in our scheme, then the computation cost is about one-eighth of that of the 160-bit ECC. To verify the actual relationship, we implemented ECDSA on Berkeley MICAz motes for various key sizes, and we measured the signature verification time.

In ECC, the signature verification time is affected by multiple factors. One of the most important factors is the curve domain parameters. When the key size is the same, different elliptic curves can lead to very different signature verification time. Finding efficient elliptic curves is an active research area, and a number of "good" curves have been found in the literature. One way to generate a "good" curve is to use the Complex Multiplication (CM) method [Atkin and Morain] to generate random elliptical domain parameters while keeping field prime p as pseudo-Mersenne primes [Konstantinou et al.]. In our implementation, we adopted this approach, i.e., we use the CM method to generate the curves with various key sizes. The signature verification time using these curves are depicted in Figure 6. We can see that the relationship roughly follows Equation (2).

It should be noted that some special curves may have smaller verification times than others. For example, the curve `secp160k1` is an elliptic curve recommended by the Standards for Efficient Cryptography Group (SECG). Its domain parameters are defined over F_p [Certicom], and it has a very small running time (12.7 seconds) in our test. Compared to this curve, the savings of the 80-bit ECC curve generated in our experiments using the CM method is 75.6%, which is quite different from the theoretical guideline (about 87.5%) derived from Equation (2); this is because the two curves are from two different curve families, and the curves generated using the CM method are not the most optimal ones.

Another important factor that can affect signature verification time is the supportive

software and hardware, such as optimization on instruction sets and specially designed hardware. The implementation by Gura et al. shows that with specialized optimization for Mica2, the 160-bit ECC signature verification can be greatly reduced to 1.6 second [Gura et al.]. Because we could not get its details to optimize the implementation for shorter keys, the results depicted in Figure 6 have not taken this kind optimization into consideration. However, projecting from the results by Gura et al. based on Equation (2), we reasonably believe that 80-bit ECC signature verification can be reduced to 0.2 second. With more research on the optimization of executing elliptic curve operations, the 80-bit ECC signature verification time will be further reduced. In the cases where this delay is still not acceptable, we can further reduce the size of the public keys used in signature verification, thus reduce the time to authenticate these signatures. In that case, we should adjust the lifetime of the public keys accordingly.

5.2 Communication Cost

The ShortPK scheme can achieve significant cost reduction on computation. However, we need to make sure that such reduction is not achieved at the cost of communication. An important goal of the ShortPK scheme is to achieve a significant improvement on computation while keeping the communication cost the same or less (compared to the standard public-key scheme).

As shown in Figure 3, a broadcast packet carries the following parts: the header (4 bits), $\frac{k}{|T|}$ (on average) public keys of size L_{pk}, [3] the indices of the public keys (8 bits each), the message M, the signature (in ECC, the length of the signature is $2 \cdot L_{pk}$ [Washington]), and the decryption key of size $L_{decrypt}$. Because the message M is the actual payload, we exclude it from the overhead calculation. We calculate the average number of bits for each broadcast packet (excluding M), and use the result as the measure for communication cost. We assume that ECDSA algorithm is used:

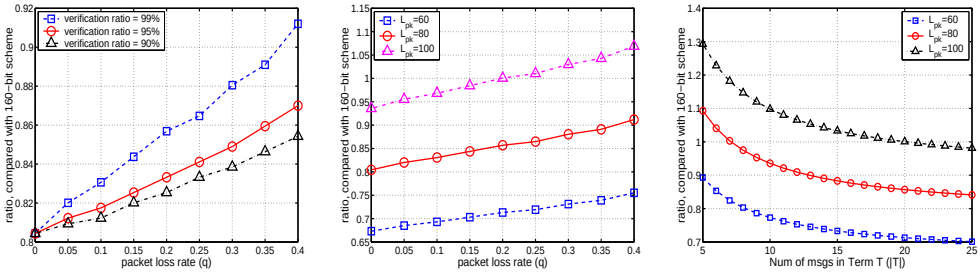
$$\text{Communication Cost} = \frac{k}{|T|} \cdot (2L_{pk} + 8) + 2L_{pk} + L_{decrypt} + 4. \tag{3}$$

Notice that loose synchronization is assumed in our scheme, we need to consider the communication cost associated with the synchronization messages. Compared with the broadcast messages, however, the synchronization is not performed very frequently [Sichitiu and Veerarittiphan]. So in analyzing the effect of the synchronization costs to the ShortPK scheme, we can amortize the synchronization cost to the broadcast messages. If we use E_{sync} to refer to the added synchronization cost, and T_{sync} to refer to the time interval during which time synchronization is performed, then for each broadcast message, the added communication cost would be

$$\text{Added Sync Cost} = \frac{\frac{E_{sync} \cdot T \cdot n}{T_{sync}}}{|T| \cdot n} = \frac{E_{sync}}{|T|} \cdot \frac{T}{T_{sync}} \tag{4}$$

In [Dai and Han], Dai and Han proposed an efficient Hierarchy Referencing Time Synchronization Protocol, in which 20 messages are necessary to maintain accurate time synchronization for a typical network where each node has 6 neighbors. If we assume this scheme is used, and the synchronization packet is 8 bytes each, the added communication

[3] Note that each public key contains both X and Y, so the size of a public key is $2L_{pk}$.



(a) Communication cost versus packet loss rate ($|T| = 20$, $T = 60$, and $L_{pk} = 80$).

(b) Communication cost versus packet loss rate ($|T| = 20$, $T = 60$, and verification ratio = 99%).

(c) Communication cost versus $|T|$ (verification ratio = 99%).

Fig. 7. Communication costs versus packet loss rate and $|T|$.

cost would be $8 \times 8 \times 20 = 1280$ bit. If the synchronization is performed each 8 hours, then the amortized cost of each message is $\frac{1280}{|T|} \cdot \frac{T}{8\times 60} = \frac{8}{3} \cdot \frac{T}{|T|}$. So we modify Equation (3) to the following:

$$\text{Communication Cost} = \frac{k}{|T|} \cdot (2L_{pk} + 8) + 2L_{pk} + L_{decrypt} + 4 + \frac{8}{3} \cdot \frac{T}{|T|}. \qquad (5)$$

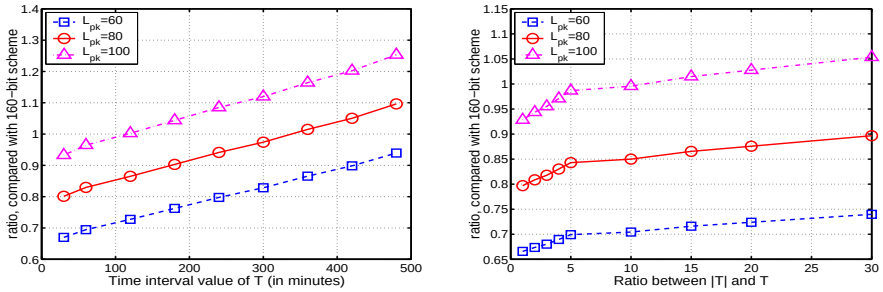
In the rest of this section, we study how packet loss rate q, T, $|T|$, and L_{pk} affect the communication costs. Since our goal is to compare with the 160-bit ECC scheme, we measure the communication cost ratio of our scheme to that of the 160-bit ECC scheme.

Packet Loss Rate q In the ShortPK scheme, k measures the degree of redundancy, and represents the number of times each public key is sent. The higher k is, the better the verification ratio, but at the same time, the communication cost becomes higher too. We study the communication cost for various packet loss rate q. Figure 7(a) and 7(b) depict the relationship between the communication cost and the packet loss rate when we fix $|T|$, L_{pk}, T, and verification ratio.

The figures show that when q becomes larger, communication costs become larger. The reason is simple: when q turns larger, to maintain the same verification ratio, each message needs to carry more public keys for the next phase, i.e., the value of k is larger; this causes higher communication overhead. The figures also show that when $L_{pk} = 80$, $T = 60$ **minutes**, and $|T| = 20$, even if the packet loss rate is as high as 40%, the ShortPK scheme still achieves a saving between 12% to 18%. This does show that the savings on computation cost is not achieved by sacrificing the communication cost.

The value of $|T|$ In the ShortPK scheme, during each short term, the base stations broadcast $|T|$ messages, all using the same public key; during the same term, k next-phase public keys are piggy-backed in these $|T|$ broadcast messages. On average, each broadcast message carries $\frac{k}{|T|}$ public keys. Therefore, when $|T|$ is larger, the average number of public keys carried by each broadcast message is lower, and hence the communication overhead becomes lower. We plot the results in figure 7(c).

From the figure, we can see that when $|T|$ is too small (e.g. $|T| = 5$), to achieve 99%



(a) Communication cost versus T (Fixed $|T| = 20$, and verification ratio = 99%).

(b) Communication cost versus $|T|/T$ (Verification ratio = 99%).

Fig. 8. Communication costs versus T and $|T|/T$.

verification ratio, the communication cost is a little bit more than the 160-bit ECDSA for $L_{pk} = 80$. However, as the frequency of broadcasting increases, the communication cost of ShortPK decreases and drops below the standard 160-bit ECDSA.

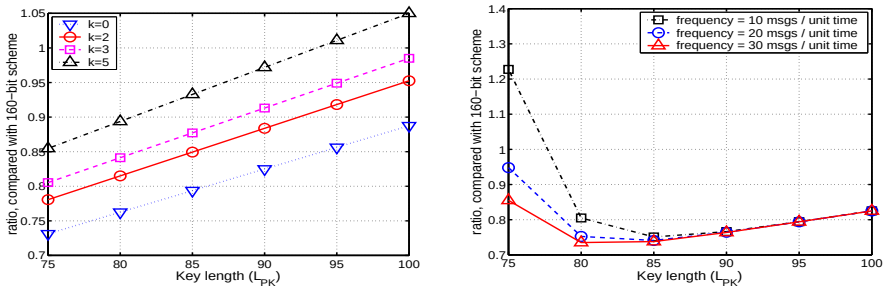
The value of $|T|$ is decided by several factors. First, it is decided by the frequency of broadcasting in applications. Second, it is affected by the size of short-term public key. If the key is longer, the security is better, and therefore, the key can be used for longer. For a fixed frequency, this means that the number of broadcasting messages (i.e., $|T|$) becomes larger during each short term. Therefore, although the communication cost for $L_{pk} = 100$ is larger than the standard ECDSA when $|T|$ is small, in practice, due to the security strength of $L_{pk} = 100$, we can use a longer term, i.e., we can use a larger $|T|$, which can reduce communication cost.

The lifetime duration of T Time synchronization will introduce additional communication cost to the proposed ShortPK scheme. Because the synchronization is not performed frequently, the added cost can be amortized by the frequent broadcast messages. The amortized cost on each broadcast message depends on several factors: the length of T, the frequency the synchronization is processed, and the number of messages broadcasted during T, i.e., $|T|$. When the frequency of the synchronization is fixed, the values of T and $|T|$ will affect the performance of ShortPK. The results are shown in figure 8

Figure 8(a) illustrates the impact of T when we fix the number of messages broadcasted in T. We make sure that the message verification ratio is 99%. Because we fix the frequency of time synchronization, the larger the value of T, the smaller number of messages broadcasted between the two synchronizations. As a result, each broadcast message will consume a larger portion of energy to perform time synchronization. We can see from the figure that for this scenario, the lifetime of public keys should be reduced.

We are more interested in the impact of ratio between $|T|$ and T on the communication cost. The result is shown in Figure 8(b). Again, we make sure that the message verification ratio is 99%. From this figure, we can see that when there are more messages during T, i.e., the ratio between $|T|$ and T is large, each broadcast message will consume relatively smaller portion of its energy on time synchronization.

The Length of Public Keys L_{pk} In addition to affecting computation cost and security



(a) Communication cost versus L_{pk} (Fixed $|T| = 20$, $|LT| = 60$).

(b) Communication cost versus L_{pk} when $k = 2$ ($|T|$ is adjusted based on L_{pk}).

Fig. 9. Communication costs versus Key Length L_{pk}.

strength, the length of public keys also affects communication cost. We plot the relationship between communication cost and key length in Figures 9(a) and 9(b).

Figure 9(a) is plotted when we fix the value of $|T| = 20$; namely, we always assume that each term lasts for the same amount of time for different L_{pk} values. It is not surprising to see from this figure that communication cost increases when the length of public key increases. However, we know that when a public key becomes longer, its security becomes stronger, so the lifetime of the key can be increased accordingly. Figure 9(b) is used to study this observation. We define the frequency of broadcasting as the number of messages broadcasted within a time unit. We use the term for $L_{pk} = 80$ as our time unit, and thus the value $|T|$ for $L_{pk} = 80$ equals the broadcast frequency. We use the security strength and frequency of $L_{pk} = 80$ as our baseline. When we change L_{pk}, to maintain the same security strength and broadcast frequency, we use Equation (1) to adjust the value of $|T|$. The results are plotted in Figure 9(b). The curves reach an interesting lowest point between $L_{pk} = 80$ and $L_{pk} = 85$ for the given frequency values. Therefore, for a given application (i.e., the frequency is fixed), if we want to maintain the pre-determined level of security, we can find the optimal key length.

6. CONCLUSION & FUTURE WORK

Broadcast authentication is a very important issue in sensor networks. With the "expensive" public key operations finding their way into sensor networks, the problem of how to save energy of the sensor nodes becomes imminent. In this paper, we present an efficient public-key-based broadcasting authentication scheme that can save energy on sensor nodes. Unlike the existing public-key-based authentication schemes that use a single strong public key, our scheme uses many weaker public keys (i.e., keys with shorter length). The security strength of our scheme is achieved by limiting the lifetime of each public key. The main challenge of our scheme is how to distribute these public keys. We have proposed a progressive distribution scheme to distribute these public keys in a way that is secure and resilient to packet loss. Compared to the standard public key authentication scheme that uses strong keys, our scheme significantly reduces the energy cost on signature verification.

Given the length of the public key and similar hardware environment, the effectiveness of our scheme depends on how fast the signature can be verified using ECDSA. This involves

the choice of optimized curve domain parameters. Although many research groups are actively studying various optimization methods, they mostly focus on stronger ECC keys; optimal domain parameters for shorter ECC keys are still unavailable. In our future work, we will use the methods developed from those studies to find better curves for shorter ECC keys.

REFERENCES

ATKIN, A. O. L. AND MORAIN, F. 1993. Elliptic curves and primality proving. *Mathematics of Computation*, 29–67.

BLOMER, J. AND MAY, A. 2001. Low secret exponent RSA revisited. *In Cryptography and Lattices - Proceedings of CALC '01 2146*, 4–19.

BONEH, D. AND DURFEE, G. 1999. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *Lecture Notes in Computer Science 1592*, 1–9.

CANETTI, R., GARAY, J., ITKIS, G., MICCIANCIO, D., NAOR, M., AND PINKAS, B. 1999. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of the 18th IEEE Conference on Computer Communications(Infocom '99)*. New York, NY.

CERTICOM, R. Certicom ecc challenge. ECC challenge website, available at http://www.certicom.com/index.php?action=res,ecc_challenge.

CERTICOM, R. 2000. Standards for efficient cryptography - SEC2: Recommended elliptic curve domain parameters. Website of SEC group, available at http://www.secg.org/download/aid-386/sec2_final.pdf.

DAI, H. AND HAN, R. 2004. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review 8*, 1, 125–139.

DU, W., , AND NING, R. W. P. 2005. An efficient scheme for authenticating public keys in sensor networks. In *In Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc*. Urbana-Champaign, Illinois, USA.

EBERLE, H., GURA, N., CHANG, S., GUPTA, V., RARICK, L., AND SUNDARAM, S. 2004. A public-key cryptographic processor for RSA and ECC.

GENNARO, R. AND ROHATGI, P. 1997. How to sign digital streams. In *Technical report, IBM T.J.Watson Research Center*.

GOLLE, P. AND MODADUGU, N. 2001. Authenticating streamed data in the presence of random packet loss. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '01)*. San Diego, CA.

GURA, N., PATEL, A., WANDER, A., EBERLE, H., AND SHANTZ, S. C. 2004. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES '04)*. Cambridge, MA, USA.

HALL, C., WAGNER, D., KELSEY, J., AND SCHNEIER, B. 1998. Building PRFs from PRPs. *Lecture Notes in Computer Science 1462*, 370–389.

HAVINGA, P. 1999. Energy efficiency of error correction on wireless systems. In *Proceedings of the IEEE Wireless Communications and Networking Conference*.

INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom)*. Boston, Massachusetts, United States.

J.M.POLLARD. 1978. Monte carlo methods for index computation (mod p). *Math. Comp. 32(143)*, 918–924.

KONSTANTINOU, E., STAMATIOU, Y., AND ZAROLIAGIS, C. 2002. On the efficient generation of elliptic curves over prime fields. *Lecture Notes in Computer Science 2523*, 333–348.

LUBY, M. AND RACKOFF, C. 1988. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing 17*, 2, 373–386.

MENEZES, A., OKAMOTO, T., AND VANSTONE, A. 1993. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transaction of Information Theory Vol 39, Issue 5*, 1639–1646.

MERKLE, R. 1980. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*.

MINER, S. AND STADDON, J. 2001. Graph-based authentication of digital streams. In *Proceedings of IEEE Symposium on Security and Privacy (IEEE S&P '01)*. Oakland, CA, 232–246.

NAOR, M. AND REINGOLD, O. 1999. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology: the journal of the International Association for Cryptologic Research 12*, 1, 29–66.

NING, P. 2006. Tinyecc: Elliptic Curve Cryptography for Sensor Networks. Cyber Defense Lab, North Carolina State University, available at http://discovery.csc.ncsu.edu/ pning/software/TinyECC/index.html.

OSTROVSKY, R. AND PATT-SHAMIR, B. 1999. Optimal and efficient clock synchronization under drifting clocks. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC)*. Atlanta, GA, United States.

PERRIG, A. 2001. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS '01)*. Philadelphia PA, USA, 28–37.

PERRIG, A., CANETTI, R., SONG, D., AND TYGAR, D. 2001. Efficient and secure source authentication for multicast. In *Proceedings of Network and Distributed System Security Symposium (NDSS '01)*. San Diego, CA.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLAR, D., AND TYGAR, J. D. 2001. Spins: Security protocols for sensor networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom '01)*. Rome, Italy, 189–199.

PUHRINGER, C. 2005. High speed elliptic curve cryptography processor for $gf(p)$. http://jce.iaik.tugraz.at/sic/content/download/713/5324/file/2005-ECC-Puehringer.pdf.

ROHATGI, P. 1999. A compact and fast hybrid signature scheme for multicast packet. In *Proceedings of 6th Conference on Computer and Communications Security (CCS '99)*. Singapore, 93–100.

SHANK, D. 1971. Class number, a theory of factorization, and genera. *Proc. Sympos. Pure Math. XX*, 415–440.

SICHITIU, M. AND VEERARITTIPHAN, C. 2003. Simple, accurate time synchronization for wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*. New Orleans, LA, United States.

SONG, D., ZUCKERMAN, D., AND J.D.TYGAR. 2002. Expander graphs for digital stream authentication and robust overlay networks. In *Proceedings of IEEE Symposium of Security and Privacy (IEEE S&P '02)*. Oakland, CA.

VANSTONE, S. 2003. Next generation security for wireless: Elliptic Curve Cryptography. *Computers and Security Vol 22, Issue 5*, 412–459.

WANDER, A., GURA, N., EBERLE, H., GUPTA, V., AND SHANTZ, S. 2005. Energy analysis of public-key cryptography for wireless sensor network. In *Proceedings of the 3rd Annual IEEE International Conference on Pervasive Computing and Communication (Percom '05)*. Hawaii, USA.

WANG, X. AND YU, H. 2005. How to break md5 and other hash functions. In *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2005)*. Aarhus, Denmark.

WANG, X., YU, H., AND YIN, Y. 2005a. Efficient collision search attacks on sha-0. In *Proceedings of 25th Annual International Cryptology Conference (CRYPTO 2005*. Santa Barbara, California, USA.

WANG, X., YU, H., AND YIN, Y. 2005b. Finding collisions in the full sha-1. In *Proceedings of 25th Annual International Cryptology Conference (CRYPTO 2005*. Santa Barbara, California, USA.

WASHINGTON, L. 2003. *Elliptic Curves: Number Theory and Cryptography*. Chapman and Hall/CRC, Inc.

WIENER, M. 1990. Cryptanalysis of short RSA secret exponents. *IEEE Transactions On Information Theory 36*, 553–558.

WOLKERSTORFER, J. 2004. Hardware aspects of elliptic curve cryptography.

WONG, C. AND LAM, S. 1998. Digital signatures for flows and multicasts. In *Proceedings of the 6th IEEE International Conference on Network Protocols (ICNP '98)*. Austin, TX.

YU, Y. AND PRANSANNA, V. 2005. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *Mobile Networks and Applications 10*, 115–131.

APPENDIX

A. PROOF OF THEOREM 3.1

PROOF. To make the notation for subscripts less confusing, we prove the following equivalent equation:

$$P_{i+1} = 1 - \{(1 - P_i) + P_i \cdot [q + (1 - q)(1 - \frac{k}{n|T|})]^{|T|}\}^n,$$

instead of its original form: $P_i = 1 - \{(1 - P_{i-1}) + P_{i-1} \cdot [q + (1 - q)(1 - \frac{k}{n|T|})]^{|T|}\}^n$.

We use $PK \rightarrow Ph_i$ to denote the event that the public key PK is accepted during phase Ph_i (note that PK is used in phase Ph_{i+1}). The probability of such an event is denoted as $\Pr(PK \rightarrow Ph_i)$; the probability that such event does not occur is denoted as $\Pr(\neg(PK \rightarrow Ph_i))$. Similarly, we define $PK \rightarrow T$ as the event that PK is accepted during the term T, where T is a term in phase i. Because there are n terms in a phase, we have the following relationship:

$$\Pr(\neg(PK \rightarrow Ph_i)) \;=\; \Pr(\neg(PK \rightarrow T))^n. \tag{6}$$

Let PK_T represent the public key used during the term T of phase Ph_i. Since PK_T is broadcasted during phase Ph_{i-1}, $\Pr(PK_T \rightarrow Ph_{i-1})$ represents the probability that PK_T is accepted into memory during phase Ph_{i-1}. As we know, this probability is P_i. In other words, during the term T, the probability that a sensor is able to verify signatures is P_i.

We now compute $\Pr(\neg(PK \rightarrow T))$, the probability that PK is not accepted during term T. For the event $\neg(PK \rightarrow T)$ to occur, there are two possibilities. First, PK_T is not in the memory, i.e., $\Pr(\neg(PK \rightarrow T) \mid \neg(PK_T \rightarrow Ph_{i-1})) = 1$. Second, PK_T is in the memory, but the packets sent by the base station during T do not cause $PK \rightarrow T$ to occur (due to packet loss or that no packet actually carries PK). Therefore, we have the following:

$$\begin{aligned}
\Pr(\neg(PK \rightarrow T)) \;&=\; \Pr(\neg(PK \rightarrow T) \mid \neg(PK_T \rightarrow Ph_{i-1})) * \Pr(\neg(PK_T \rightarrow Ph_{i-1})) \\
&\quad + \Pr(\neg(PK \rightarrow T) \mid (PK_T \rightarrow Ph_{i-1})) * \Pr(PK_T \rightarrow Ph_{i-1}) \\
&= \Pr(\neg(PK_T \rightarrow Ph_{i-1}) + \Pr(\neg(PK \rightarrow T) \\
&\qquad \mid (PK_T \rightarrow Ph_{i-1})) * \Pr(PK_T \rightarrow Ph_{i-1}) \\
&= (1 - P_i) + P_i * \Pr(\neg(PK \rightarrow T) \mid (PK_T \rightarrow Ph_{i-1})).
\end{aligned} \tag{7}$$

Next, we compute $\Pr(\neg(PK \rightarrow T) \mid (PK_T \rightarrow Ph_{i-1}))$. A term T has $|T|$ broadcast messages, we divide T into $|T|$ time points, at each of which one broadcast message is sent. We use t to represent any one of these time points, and define $PK \rightarrow t$ as the event that PK is accepted at the point t. We first compute $\Pr(\neg(PK \rightarrow t) \mid (PK_T \rightarrow Ph_{i-1}))$, the probability that PK is not accepted at time t when PK_T is in the memory. In the following discussion, for the sake of simplicity, we omit the condition $(PK_T \rightarrow Ph_{i-1})$ from the expressions.

$$\begin{aligned}
&\Pr(\neg(PK \rightarrow t) \mid (PK_T \rightarrow Ph_{i-1})) \\
=\; &\Pr(\text{“Packet is lost”}) + \Pr(\text{“Packet is not lost”}) \cdot \Pr(\text{“Packet does not carry PK”}) \\
=\; &q + (1 - q)\frac{n - \frac{k}{|T|}}{n},
\end{aligned} \tag{8}$$

where $\Pr$("The packet does not carry PK") is calculated in the following way: since each packet carries $\frac{k}{|T|}$ public keys that are selected from n keys (without duplicate), the probability that these keys do not include PK is the following

$$\frac{n-1}{n} \cdot \frac{n-2}{n-1} \cdots \frac{n-\frac{k}{|T|}}{n-\frac{k}{|T|}+1} = \frac{n-\frac{k}{|T|}}{n} = 1 - \frac{k}{n|T|}.$$

Since there are $|T|$ messages within each term T, we have

$$\begin{aligned} \Pr(\neg(PK \rightarrow T) \mid (PK_T \rightarrow Ph_{i-1})) &= \Pr(\neg(PK \rightarrow t) \mid (PK_T \rightarrow Ph_{i-1}))^{|T|} \\ &= [q + (1-q)(1 - \frac{k}{n|T|})]^{|T|}. \end{aligned} \quad (9)$$

Combining Equations (6), (7), and (9) together, we have the following:

$$\begin{aligned} P_{i+1} &= \Pr(PK \rightarrow Ph_i)) = 1 - \Pr(\neg(PK \rightarrow Ph_i))^n \\ &= 1 - \{(1-P_i) + P_i \cdot [q + (1-q)(1 - \frac{k}{n|T|})]^{|T|}\}^n. \end{aligned}$$

Received February 2007; revised December 2007; accepted December 2008