An abridged version of this paper appears in the IEEE MILCOM 2001 Conference Proceedings.
(October 28 - 31, 2001, Sheraton Premiere at Tysons Corner, Vienna, Virginia, USA)

MUTUAL AUTHENTICATION AND KEY EXCHANGE FOR LOW POWER WIRELESS COMMUNICATIONS*

Duncan S. Wong and Agnes H. Chan

College of Computer Science
Northeastern University, Boston, MA 02115
{swong, ahchan}@ccs.neu.edu

ABSTRACT

We propose two efficient Mutually Authenticated Key Exchange Protocols (MAKEPs) namely (1) server-specific and (2) linear MAKEPs. They are suitable for establishing secure communications between a low-power wireless device (client) and a powerful base station (server) under different system requirements. These protocols reduce the computational burden of the client while still maintaining a required level of security and a good amount of scalability for most wireless communications applications. We discuss their security and also demonstrate that they can be implemented efficiently.

I INTRODUCTION

Low-power wireless devices, such as Personal Digital Assistants (PDAs) and cellular phones, are characterized by their limited memory capacity, low computational power, small and monochrome screens. In addition, the wireless environment is limited in bandwidth and is subject to erratic changes such as weather, terrain and external interferences. These constraints have prevented a simple migration of cryptographic protocols that have been widely adopted in wire-line networks to wireless networks for authentication and security. Even worse, the wireless communications medium is also more difficult to be protected against active attacks and eavesdropping. An eavesdropper can 'tap' into a communication channel over-the-air without being detected. Besides, an adversary can also jam a channel easily. Furthermore, due to the mobility of wireless devices, authentication of both wireless devices and base stations becomes paramount important. Therefore, a set of protocols needs to be designed for these low-power wireless devices so that mutual authentication and data privacy can be accomplished efficiently and effectively in terms of computational complexity, memory demand and bandwidth requirement. In general, they should be light-weighted, requiring minimal user interventions, simple to implement and fast but also maintaining an expected level of security and a certain amount of scalability.

In the current cellular networks [13, 11], authentication consists of verifying the authenticity of a mobile unit. This helps prevent phone cloning fraud and the abuse of usage by any unregistered mobile units. The mobile, on the other hand, would not authenticate a serving system because it is not the interest of the mobile as long as the serving system is providing the expected service to it, that is, transporting real-time voice communications between the mobile and some intended parties. For the next generation of cellular networks however, the serving system will also provide secure data communications service and be able to upgrade or change the data stored in the mobile unit under permission. These services definitely require the mobile unit to authenticate the serving system. Hence the authentication of serving systems will be in a prominent position in the next generation of cellular networks.

For public-key based MAKEPs [1, 8], their computational complexities are usually very high. As shown in [6], three to four minutes of pure computation time is required to generate a 512-bit RSA key pair on a 16MHz Palm Pilot and more than 7 seconds of CPU time is needed to generate a signature.

*This work was sponsored by the U. S. Air Force under contract F30602-00-2-0518

Such low speed is usually unacceptable in practice and prevents the public key cryptosystems being widely deployed onto most of the applications running on low-power wireless devices. This drawback is more noticeable when they are compared with symmetric key based protocols [12, 3, 2]. As we will see in Section V, the symmetric key algorithms are much faster than those public-key algorithms when implemented on Palm devices. Therefore in many well-known wireless systems such as [9, 7], a simple challenge-response scheme with a shared secret key has now become the de facto standard for authentication. However symmetric key based schemes either need the two communicating parties to share a long-live key or have a third party involved during the runtime. In the first case, each party has to maintain a set of distinct keys for communicating with different partners. In the later case, a centralized trusted party is needed whenever the protocol is executed. Hence the key management and scalability are two major issues when deploying symmetric key based schemes.

In this paper, we propose and analyze two efficient MAKEPs for wireless communications between a low-power wireless device (client) and a powerful base station (server). By referring a powerful base station, we mean a machine which has similar computational power to a current desktop computer. We will also discuss their performance and implementation efficiency. In the next section, we start with some background and notations used in the paper, including a general protocol which highlights the major elements of a pure public key based MAKEP. The protocol will be used as a basis for comparison with our protocols in terms of computational complexity.

II PRELIMINARIES

Let $\mathcal{E}_K$ and $\mathcal{E}_K^{-1}$ denote the encryption and decryption transformations under the symmetric key K respectively. For public key systems, each entity A has a public key PK_A and a corresponding private key SK_A. For simplicity of notation, we use $\mathcal{E}_{PK_A}$ and $\mathcal{E}_{SK_A}$ respectively to denote public key encryption and decryption. The subscript of $\mathcal{E}$ represents the key which makes the cryptographic function self-explained. Sig_{TA} is a *secret* signing algorithm and Ver_{TA} is the corresponding *public* verification algorithm of a trusted authority TA. A certificate $Cert_A$ of an entity A is denoted by

$$Cert_A = < \mathrm{ID}_A, m, Sig_{\mathrm{TA}}(\mathrm{ID}_A, m) >,$$

where ID_A is the identification information of A and m is some message certified by TA that binds to ID_A. By $< m >$, we mean an appropriate encoding of m. A *nonce*, denoted by $r \leftarrow \{0,1\}^l$, is a l-bit random number generated by some cryptographically strong random number generator and each value of r will be used only once. We use k to represent a security parameter.

A		**B**
(PK_A, SK_A)		(PK_B, SK_B)
$r_A \leftarrow \{0,1\}^{f(k)}$		
	$\xrightarrow{r_A,\ Cert_A = < \mathrm{ID}_A, PK_A, Sig_{\mathrm{TA}}(\mathrm{ID}_A, PK_A) >}$	
		$c_B \leftarrow \{0,1\}^k$
		$\beta = \mathcal{E}_{PK_A}(c_B)$
		$S_B = \mathcal{E}_{SK_B}(\mathrm{ID}_A, r_A, \beta)$
		$r_B \leftarrow \{0,1\}^{f(k)}$
	$r_B,\ \beta,\ S_B,\ Cert_B$ where $\xleftarrow{Cert_B = < \mathrm{ID}_B, PK_B, Sig_{\mathrm{TA}}(\mathrm{ID}_B, PK_B) >}$	
$c_A \leftarrow \{0,1\}^k$		
$\alpha = \mathcal{E}_{PK_B}(c_A)$		
$S_A = \mathcal{E}_{SK_A}(\mathrm{ID}_B, r_B, \alpha)$		
	$\xrightarrow{\alpha,\ S_A}$	
$\sigma = c_A \oplus c_B$		$\sigma = c_A \oplus c_B$

Figure 1: Public Key Based Mutual Authentication and Key Exchange Abstract Protocol

We will now identify the primitive elements of the public key MAKEPs given in [1, 8], and generalize to an abstract protocol as shown in Figure 1. The abstract protocol shows two primitive elements. The first one is the two challenge-response pairs denoted by (r_A, S_B) and (r_B, S_A). They provide the mutual authentication. The second primitive element is the 'binding' of each party's encrypted session key contribution (α or β) to the corresponding challenge number (r_B or r_A). These bindings (S_A and S_B) are used to ensure the *freshness* of the session key. This technique helps detect both replay attack and man-in-the-middle attack.

III SERVER-SPECIFIC MAKEP

During the execution of a public-key based MAKEP, both parties A and B are required to carry out public key encryption and decryption computations, as well as signature verification. These operations can be very inefficient when implemented on low-power wireless devices. In this section, we propose a new MAKEP which significantly reduces the computational requirement of a low-power client when it is communicating with a powerful server. This protocol eliminates any usage of public-key cryptographic operations at the client side and replaces them with efficient symmetric key based operations. The protocol is shown in Figure 2.

$$\begin{array}{lcr} \textbf{A} & & \textbf{B} \\ K_A & & (PK_B, SK_B) \\ r_A \leftarrow \{0,1\}^k & & \\ & \xrightarrow{\quad \mathcal{E}_{K_A}(r_A),\ Cert_A^B \quad} & \\ & & r_B \leftarrow \{0,1\}^k \\ & \xleftarrow{\quad \mathcal{E}_{K_A}(r_A, r_B, \mathrm{ID}_B) \quad} & \\ & \xrightarrow{\quad \mathcal{E}_{K_A}(r_B) \quad} & \\ \sigma = r_A \oplus r_B & & \sigma = r_A \oplus r_B \end{array}$$

Figure 2: Server-specific MAKEP

Before running the protocol with a server B, the client A first obtains a certificate from TA, given by

$$Cert_A^B = < \mathrm{ID}_A, \mathcal{E}_{PK_B}(K_A), Sig_{\mathrm{TA}}(ID_A, \mathcal{E}_{PK_B}(K_A)) >$$

where K_A is A's *long-live symmetric key*. Inside the certificate, K_A is encrypted under B's public key PK_B. This requires A to possess a certificate specific to server B before communicating with B. In other words, each certificate is *server-specific*. We hereby assume that A and TA have authentic copies of PK_B whenever needed.

The protocol starts when A chooses a nonce r_A as a challenge to authenticate B and encrypts it under K_A. She sends $\mathcal{E}_{K_A}(r_A)$ and her certificate, $Cert_A^B$, to B. When B receives the message from A, he checks the certificate by running TA's public verification algorithm Ver_{TA}. If the certificate is valid, the protocol proceeds; otherwise, it halts. B decrypts $\mathcal{E}_{PK_B}(K_A)$ using his private key SK_B and obtains K_A. Now K_A is the shared secret key of A and B. B chooses a nonce r_B and encrypts r_A,r_B and his identifier ID_B under K_A. The encrypted message, $\mathcal{E}_{K_A}(r_A, r_B, \mathrm{ID}_B)$, is then sent back to A. A decrypts it and checks if the message contains r_A, ID_B and some other number. If this is true, the other number is said to be r_B. A knows that B sent this message since, apart from A, only B knows K_A. This completes the authentication of B. A then encrypts r_B and sends it back to B. In the meantime, A also computes the session key σ as $r_A \oplus r_B$. When B receives $\mathcal{E}_{K_A}(r_B)$, he decrypts it and checks if it is r_B. If it is true, the authentication of A is completed. B then computes the session key accordingly. In practice, it is important to have the symmetric-key encryption algorithm be used in a way that it can prevent block replay attack [4].

We observe that both A and B supply their contributions r_A and r_B respectively. In this way, no single party has the full control on the selection of the session key, and both A and B can ensure the freshness of the final session key.

If there are n distinct servers in the system that A wants to communicate with, she needs n distinct certificates, one for each server. Furthermore, the long-live key of A for each server needs to be distinct from her keys for other servers in the system; otherwise, a server could impersonate a client and authenticate himself to *other* servers. This requirement is needed only if an application in practice has this kind of concerns. Since B shares K_A with A after receiving $Cert_A^B$, if B colludes with an adversary C, C can get K_A and impersonate A. Thus, this scheme cannot prevent a server to impersonate its *own* clients. In practice, this may be acceptable to most of the applications. In fact this characteristic is shared by all protocols in the pure symmetric key authentication and key distribution systems [12, 3, 2].

Security This scheme can be shown to be secure under a security model similar to that used by Bellare and Rogaway in [2]. We observe that the server authentication is accomplished by the challenge-response pair $(\mathcal{E}_{K_A}(r_A),\ \mathcal{E}_{K_A}(r_A, r_B, \mathrm{ID}_B))$. It is infeasible for an adversary to compute the response without knowing K_A, thus A can be sure that only B can produce the response. The client authentication is done in a similar way. In addition, it is not difficult to see that an adversary cannot obtain any information of r_A and r_B if the encryption scheme is secure, which implies that the session key contribution of each party is transferred to the counter-party securely. A complete analysis will be given in the full paper.

Key Management and Scalability If we remove $Cert_A^B$ from the first message and assume that B knows K_A before executing the protocol with A, then this scheme is no different from a conventional symmetric key based scheme. However, in a conventional symmetric key based scheme, the server B has to maintain a secure database of all its clients' long-live keys. In the proposed scheme, the cost of maintaining and searching through such a database is eliminated. Instead, each client keeps a certified long-live key and sends it securely to the server whenever the protocol is executed. Furthermore, A can change her key anytime by obtaining a new certificate from TA without involving B. Therefore there is no key synchronization problem between A and B. The only extra cost incurred in the proposed scheme is the memory space required to store the certificates. For applications where the number of servers that each client needs to communicate with is relatively small with respect to the memory capacity of the client, this scheme provides a good key management solution in general and is scalable.

In practice, if k, K_A and ID_A are each 128 bits long and 512-bit RSA is used for public key cryptographic operations, we only need 160 bytes of memory to store a long-live key and a certificate. Thus less than 8KB of memory is required if A wants to communicate with 50 different servers. The performance and implementation efficiency of this protocol will be compared with other protocols in Section V.

Further Enhancements and Variations To further reduce the power consumption incurred by client A, we can decrease the number of transmissions of the certificate $Cert_A^B$. Since the server B knows the long-live symmetric key K_A after receiving the first $Cert_A^B$, B can cache K_A for multi-sessions if A uses the same K_A for establishing more than one session within a short period. Here we are assuming that the cache can be kept relatively secure within the short period.

Another variation of the protocol is to send $h(r_B)$ instead of $\mathcal{E}_{K_A}(r_B)$ in the third message, where h denotes a one-way hash function. This variation can be more efficient if there exists a one-way hash function which is faster than the symmetric key ciphers available in a system or the length of the digest is shorter than the ciphertext produced from the encryption of r_B.

To counteract reflection attack [10], the response from B to A's challenge includes B's own identifier. An alternative solution is to use distinct keys K_A and K'_A for encryptions from A to B and B to A respectively. In general, the attack will not be of concern if each party can either be an initiator or a responder but not both in the system.

IV LINEAR MAKEP

In this section, we present another protocol called Linear MAKEP. In general, it allows each client to communicate with as many servers as it wants without inducing any scalability problems. In addition, it also prevents any server to impersonate its own clients although this may not be essential in most applications. The protocol is illustrated in Figure 3.

A		**B**
$\{a_1, \cdots, a_{2i-1}, a_{2i}, \cdots, a_{2n}\} \in_R \mathbb{Z}_{p-1}$		(PK_B, SK_B)
$\{g^{a_1}, \cdots, g^{a_{2i-1}}, g^{a_{2i}}, \cdots, g^{a_{2n}}\} \in \mathbb{Z}_p^*$		
	$\xrightarrow{Cert_A^i}$	
		$r_B \in_R \mathbb{Z}_{p-1}$
	$\xleftarrow{r_B}$	
$r_A \leftarrow \{0,1\}^k$		
$x = \mathcal{E}_{PK_B}(r_A)$		
$y = a_{2i-1}(x \oplus r_B) + a_{2i} \bmod (p-1)$		
	$\xrightarrow{x, y}$	
$\sigma = r_A \oplus y$		$(g^{a_{2i-1}})^{x \oplus r_B} g^{a_{2i}} \stackrel{?}{\equiv} g^y \pmod p$
		$\sigma = r_A \oplus y$
	$\xleftarrow{\mathcal{E}_\sigma(x)}$	

Figure 3: Linear MAKEP

Let p be a prime such that the discrete logarithm problem in $\mathbb{Z}_p$ is intractable. Let $g \in \mathbb{Z}_p^*$ be a primitive element. A randomly chooses a sequence of integers $(a_1, a_2, \cdots, a_{2n})$ in $\mathbb{Z}_{p-1}$ as her secret keys. The corresponding sequence of public keys is $(g^{a_1}, g^{a_2}, \cdots, g^{a_{2n}})$ in $\mathbb{Z}_p^*$. For each pair of public keys, $(g^{a_{2i-1}}, g^{a_{2i}})$, where $1 \le i \le n$, a signature is obtained from the

TA. The signatures can be represented as a set of $\{Sig_{\mathrm{TA}}(\mathrm{ID}_A, g^{a_{2i-1}}, g^{a_{2i}})\}_{1\leq i\leq n}$.

At the i-th run of the protocol, A constructs a certificate given by

$$Cert_A^i =< \mathrm{ID}_A, g^{a_{2i-1}}, g^{a_{2i}}, Sig_{\mathrm{TA}}(\mathrm{ID}_A, g^{a_{2i-1}}, g^{a_{2i}}) >$$

and sends it to B. B then sends back a nonce r_B after verifying the validity of the certificate. When A receives it, she picks another nonce r_A and computes $x = \mathcal{E}_{PK_B}(r_A)$. She then computes y as

$$y = a_{2i-1}(x \oplus r_B) + a_{2i} \bmod (p-1) \qquad (1)$$

and sends (x, y) to B. In the meantime, she computes a new session key σ as $r_A \oplus y$. When B receives x and y, he determines whether

$$(g^{a_{2i-1}})^{x\oplus r_B} g^{a_{2i}} \stackrel{?}{\equiv} g^y \pmod{p}. \qquad (2)$$

If equation (2) holds, B decrypts x to obtain r_A, computes the new session key σ and sends $\mathcal{E}_\sigma(x)$ back to A; otherwise, B rejects the communication and the protocol halts. After A receives a message from B again, she decrypts the message and checks if the decrypted message is x. If it is correct, a new session is established. Similar to Server-specific MAKEP, B can send $h(\sigma)$ back to A instead of $\mathcal{E}_\sigma(x)$ if there exists an efficient one-way hash function, h.

The mutual authentication is accomplished by the challenge-response pairs (r_B, y) and (x, $\mathcal{E}_\sigma(x)$). Secondly, the session key r_A is conveyed securely from A to B by encrypting it under B's public key and is 'signed' by A through the signature-like mechanism of (1) and (2). It is also bound to r_B providing B the capability to check the freshness of keys.

We note that the total number of times that A can run the protocol is limited by the total number of key pairs she possesses unless she can generate more key pairs and obtain the signatures from the TA for these keys during the idle time. This feature can be used to enforce the total number of times a client can access the service provided by a particular server.

In Server-specific MAKEP, A and B are sharing a symmetric key K_A and the client authentication is no more than a conventional symmetric key based authentication scheme. Therefore, the server can 'impersonate' its own subscribers. In this scheme however, the server does not share any key with the client. In addition, any certificate can be used to communicate with any server because the certificate is not server specific. In general, the scheme can be used in a large finite group such that the discrete log problem in the group is intractable.

Security The server authentication is accomplished by the challenge-response pair $(x, \mathcal{E}_\sigma(x))$. If it is infeasible for an adversary to decrypt x without knowing SK_B, A can be sure that only B can obtain r_A from x and therefore only B can produce $\mathcal{E}_\sigma(x)$. The client authentication is done by the challenge r_B from B and the response pair (x, y) from A. Intuitively, we can consider that y is a signature on r_B. Hence any secure signature scheme should be suitable for this purpose. We use the scheme described above because it requires very little computation overhead for the client. A formal treatment will be given in the full paper.

Further Enhancements To save the memory space, we choose $g \in \mathbb{Z}_p^*$ to be a q-th root of 1 modulo p where q is a 160-bit prime and p is a 512-bit prime. To estimate the memory requirement, we assume that there are $2n$ secret keys currently stored in the client's local memory. We also assume that the TA signature algorithm is 512-bit RSA based, and the length of ID_A is 128 bits. Then the total space required can be found to be $(16 + 232n)$ bytes. Hence if A has 8KB of memory for data, she can store 35 sets of keys and signatures. If A has more memory, say 32KB, the number of key and signature sets can go up to 140.

V PERFORMANCE

In this paper, we present the protocols without specifying any particular cryptographic algorithm. The choice of which is up to the applications taking into consideration their systems' capabilities and constraints. The performance comparison given here is therefore based on the number of times the cryptographic operations have to be performed, the sizes of the messages and the total number of messages sent in each protocol. Table 1 shows the computational complexity required by the two protocols and the public-key based abstract protocol in terms of the number of symmetric key operations and the number-theoretic operations when RSA is adopted as the public key cryptographic algorithm.

For the Server-specific MAKEP, all computation-intensive public-key cryptographic operations are carried out on the server side while requiring only three symmetric key encryption/decryption and one exclusive-or operations on the client side. To encrypt one 128-bit block of data, we find that the stream cipher, SSC2 [15], takes 3.93 msec on a 16MHz Palm V. Even the ARC4 (Alleged RC4)[1] takes only 4.16 msec. For block cipher like Rijndael [5], the time taken is about 12 msec [14]. Hence these figures suggest that an efficient implementation of the scheme can be achieved easily.

For the Linear MAKEP, the computational requirement on the client is given by one public key encryption, one modular multiplication, one modular addition, and one symmetric key decryption. The more expensive operations for computing the client's keys are carried out offline before running the protocol. It can also be done by a more powerful and trusted third party whenever applicable. According to our implementation[2] of the multiple-precision integer arithmetic operations for Palm OS, we find that a 512-bit modular multiplication takes about 107 msec on a 16MHz Palm V and a modular addition takes 3.28 msec. For a 512-bit modular exponentiation, it takes about 365 msec if the exponent is 3. Hence less than 500 msec of pure computation is required for the client. Furthermore, if the server is known before running the protocol, the public key encryption carried out by the client can also be done offline in advance. Hence by using the pre-computation technique, the efficiency of the protocol can be improved to about 120 ms of pure computation at the client side.

		Mod Exp		Mod	Symmetric
		small exp	large exp	Mult	key crypto ops
Pub key based abstract prot.	A	3	2	0	0
	B	3	2	0	0
Server Specific	A	0	0	0	3
	B	1	1	0	3
Linear	A	1	0	1	1
	B	1	3	1	1

Table 1: A Comparison of the Computational Complexity of the Protocols

VI ACKNOWLEDGEMENTS

The authors would like to thank Hector Ho Fuentes and Feng Zhu for their suggestions on improving the efficiency during the preliminary work of the paper.

References

[1] Aziz, A., and Diffie, W. A secure communcations protocol to prevent unauthorized access – privacy and authentication for wireless local area networks. *IEEE Personal Communications* (First Quarter 1994).

[2] Bellare, M., and Rogaway, P. Entity authentication and key distribution. In *Proc. CRYPTO 93* (1994), D. R. Stinson, Ed., Springer, pp. 232–249. Lecture Notes in Computer Science No. 773.

[3] Bellare, M., and Rogaway, P. Provably secure session key distribution – the three party case. In *Proc. 27th ACM Symp. on Theory of Computing* (Las Vegas, 1995), ACM, pp. 57–66.

[4] Campbell, C. M. Design and specification of cryptographic capabilities. *IEEE Communications Society Magazine 16*, 6 (Nov 1978).

[5] Daemen, J., and Rijmen, V. AES Proposal: Rijndael. *AES Algorithm Submission* (Sep 1999). http://www.nist.gov/aes.

[6] Daswani, N., and Boneh, D. Experimenting with electronic commerce on the PalmPilot. *Financial Cryptography '99, Lecture Notes in Computer Science Vol. 1648* (1999).

[7] ETSI SAGE Task Force for 3GPP. *Report on the Design and Evaluation of The MILENAGE Algorithm Set: An Example Algorithm of the 3GPP Authentication and Key Generation Functions*, Dec 2000.

[8] Freier, A. O., Karlton, P., and Kocher, P. C. *The SSL Protocol Version 3.0.* INTERNET-DRAFT, Nov 1996. www.netscape.com/eng/ssl3/draft302.txt.

[9] LAN/MAN Standards Committee of the IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, 1999 ed., 1999.

[10] Mitchell, C. Limitations of challenge-response entity authentication. *Electronics Letters 25*, 17 (Aug 1989).

[11] Mouly, M., and Pautet, M.-B. *The GSM System for Mobile Communications.* Published by the authors, 1992.

[1] http://www.achtung.com/crypto/arcfour.txt
[2] www.ccs.neu.edu/home/swong/MPLib

[12] Needham, R. M., and Schroeder, M. D. Using encryption for authentication in large networks of computers. *Communications of the ACM 21*, 12 (Dec. 1978), 993–999.

[13] The Telecommunications Industry Association (TIA). *Mobile Station-Base Station Compatibility Standard for Wideband Spread Spectrum Cellular Systems (TIA/EIA-95-B-99)*, Feb 1999.

[14] Wong, D. S., Ho Fuentes, H., and Chan, A. H. The performance measurement of cryptographic primitives on palm devices. to appear in the Proceedings of the 17th Annual Computer Security Applications Conference, Dec 2001.

[15] Zhang, M., Carroll, C., and Chan, A. H. The software-oriented stream cipher SSC2. *Fast Software Encryption Workshop 2000* (2000).