# A LIGHT-WEIGHT MUTUAL AUTHENTICATION AND KEY-EXCHANGE PROTOCOL BASED ON ELLIPTICAL CURVE CRYPTOGAPHY FOR ENERGY-CONSTRAINED DEVICES

Kin Choong Yow and Amol Dabholkar
School of Computer Engineering,
Nanyang Technological University,
Singapore 639798,
email: kcyow@ntu.edu.sg, amold@pmail.ntu.edu.sg

**Abstract**

Wireless devices are characterized by low computational power and memory. Hence security protocols dealing with these devices have to be designed to give minimal computational and memory load. We present an efficient authentication and key exchange protocol for low-end wireless clients and high end servers, which is overall nearly three times as fast as comparable protocols. The basic idea of our protocol is to use symmetric key encryption in place of public key encryption wherever possible.

## 1. Introduction

Wireless environment is by its very nature insecure and limited in bandwidth [1]. For example, an adversary could easily tap into wireless communication channels or jam other people's devices. Furthermore personal wireless devices are generally of low power, low in computational abilities and low in memory. These reasons have prevented a simple migration of cryptographic protocols from fixed networks to wireless networks for authentication and security [2].

The first phase of a typical secure transaction or communication is a secure key exchange where both parties decide on a common secret-key known only to them. The security of the rest of the transaction then depends on this first step and if there is a compromise at this phase, the rest of the communication may be compromised.

In this paper we will look at protocols for key exchange and ways for devices to authenticate each other. We will be looking specifically at protocols that combine these two objectives, which are called Mutually Authenticated Key Exchange Protocols or MAKEPs. The goal of these protocols is to provide the communicating parties with some assurance that they know each others' true identity and at the same time have a common key known only to them [3].

We will first look into the design strategies for a typical MAKEP and study two recently proposed protocols that deal with similar scenarios. We will then present our protocol and discuss its design and security. Finally we will compare implementations of the protocols based on elliptic curve cryptography using PDA clients.

## 2. Design Strategies

In this section we will briefly analyze the designs of two MAKEPs designed for low-powered clients and discuss their advantages and disadvantages.

*2.1 Server-Specific MAKEP*

This protocol was proposed in [2] as an efficient MAKEP for communication between a low powered wireless client and a powerful server. The main feature of this MAKEP is that it eliminates any public key cryptographic operations on the client side. The client side performs only symmetric key operations. Symmetric key cryptography is much faster than public key cryptography [4]. Hence the protocol executes much faster on the client side, than it would have if public key operations were involved.

The protocol relies on server-specific certificates of the form given by $Cert_A^B$ = <$ID_A$, $\{K_A\}_{PKB}$, $Sig_{TA}(ID_A, \{K_A\}_{PKB})$> which is a certificate from the low powered wireless client *Alice* to the high powered server *Bob*.

Here $K_A$ is *Alice*'s long-lived symmetric secret key. $PK_B$ is the well known public key of B. $\{K_A\}_{PKB}$ means *Alice*'s secret key $K_A$ is encrypted with *Bob*'s public key $PK_B$. $Sig_{TA}$ is a signature from a trusted authority TA, and $ID_A$ is *Alice*'s identity.

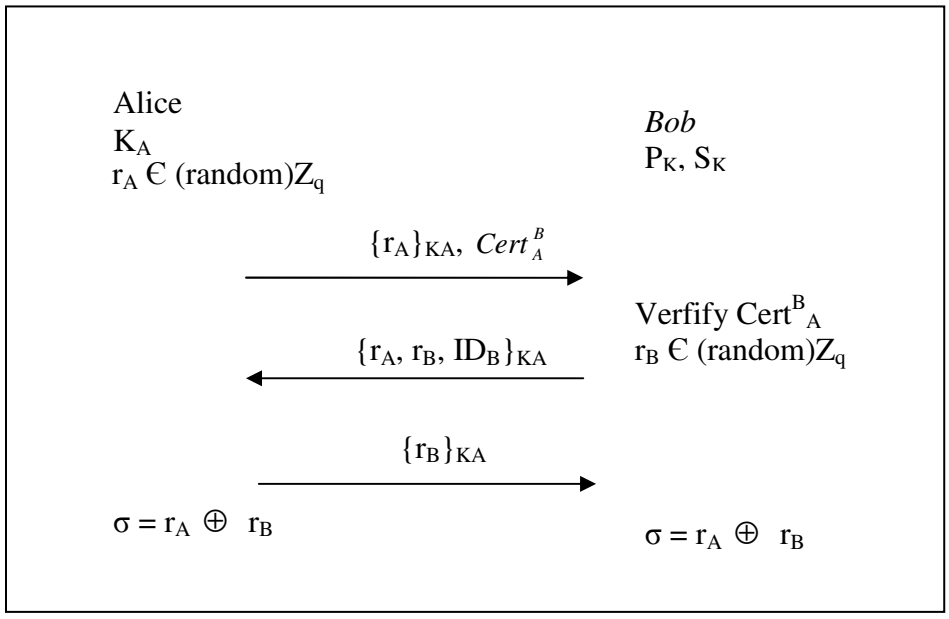


Figure 1: Server-Specific MAKEP protocol

On receiving the first message from *Alice*, *Bob* verifies the TA's signature and obtains *Alice*'s long lived symmetric key $K_A$, by decrypting $\{K_A\}_{PKB}$ from the certificate by using his private key $SK_B$. This step also authenticates *Alice*. *Bob* then encrypts a new random value $r_B$ along with *Alice*'s random value $r_A$ and his identity $ID_B$, with *Alice*'s symmetric key $K_A$. This step authenticates *Bob*. Now the session key $\sigma$ is calculated as $r_A \oplus r_B$.

The protocol completes in 3 steps with hardly any pre-computations. Another major advantage is that the client uses only symmetric-key cryptography with her secret key $K_A$. This increases its speed and efficiency as compared to other comparable MAKEPs that use public-key cryptography. The freshness of the session key is maintained by using the random numbers $r_A$ and $r_B$. Unlike the Jakobsson-

Pointcheval protocol [5], where only the client computes the session key σ, here we use client and server contributions, $r_A$ and $r_B$ respectively, to calculate σ. This increases the strength of the key.

However, there are some disadvantages to this protocol in its raw form –

1. The protocol needs server specific certificates. For *n* servers it will need n distinct certificates. This creates scalability issues.
2. The client's long lived secret key $K_A$ is known to the server. If the server is malicious it can create nonexistent sessions and there is no way for *Alice* to deny these occurred, i.e. a malicious server can impersonate its clients to create false runs of the protocol.
3. Even if the server is trustworthy, it has access to the client's secret key $K_A$. If the server's security is compromised in any way it implies the client's security is also compromised since a hacker can get $K_A$ from *Bob*'s memory.

These problems can be taken care of by our proposed protocol described in the next section.

### *2.2 Client-Server MAKEP*

This protocol solves the scalability and other problems of the Server-Specific protocol by using public-key cryptography on the client side as well as the server side [3]. However, the public-key computations are kept to a minimum and most of the costly operations are done on the server side.

*Alice* chooses a random value $r_A$ as its contribution towards creating the shared key, and encrypts it using *Bob*'s well known public key $PK_B$. *Alice* also computes a random value '*b*' and sends $\beta = g^b$ to *Bob*.

*Bob* replies by sending his shared key contribution $r_B$, encrypted with *Alice*'s contribution $r_A$. This authenticates *Bob* to *Alice*. Since $r_A$ had been encrypted under *Bob*'s private key, only *Bob* will know $r_A$, and $r_B$ encrypted under $r_A$ implies that *Bob* is replying to the current session. *Alice* sends the value $y = a\text{h}(\sigma) + b \bmod q$ ($q$ is the prime number used to define the field $F_q$) to *Bob* where σ is computed from $r_A$ and $r_B$. *Bob* checks if $g^y = (g^a)^{h(\sigma)}\beta$. Now $g^y = g^{(ah(\sigma) + b)} = (g^a)^{h(\sigma)}g^b = (g^a)^{h(\sigma)}\beta$. *Bob* knows $g^a$ from the certificate and he has received β in the previous message. Thus *Alice* is authenticated as the secret key portion of 'a' in the value of y can only be computed by *Alice*, and the β value effectively binds the earlier and later messages, as *Alice* will compute a new value of '*b*' for each session.

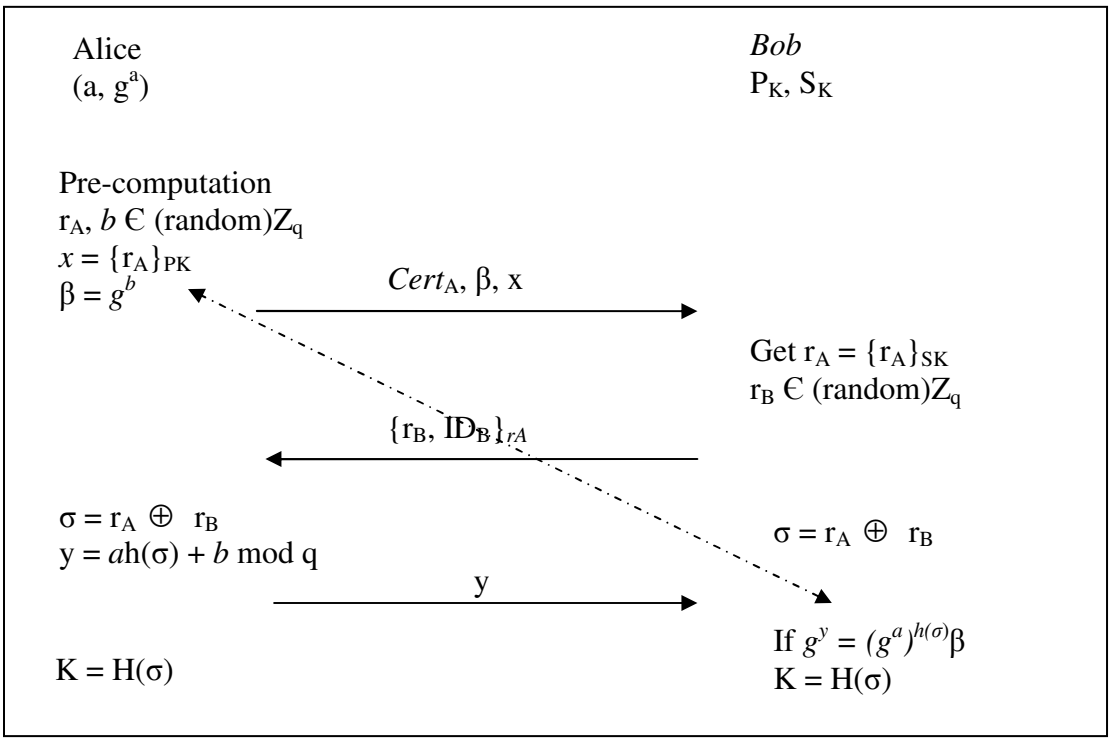


Figure 2: Client Server MAKEP

The third step is necessary because *Alice* has not been authenticated to *Bob*. Note that in the first step, when *Alice* sends "$Cert_A$, β, x" to *Bob*, this does not authenticate *Alice*, as anyone could send this message by capturing previous protocol runs. The dotted line shows the binding between the first and the third step by *b* and β. The third step combines the secret key (*a*) and the session random (*b*) value of *Alice* to authenticate her to *Bob*. If *Alice* was authenticated in the first step itself, the third step will not be necessary.

This protocol has been shown to be secure against attacks by using the Bellare-Rogaway [6] model. Since it uses general certificates there is no scaling problem like the server-specific MAKEP. Most of the computationally expensive operations like Certificate verification are carried out on the server-side.

## 3. Our Proposed Protocol

We observe that by using the keys of the client and the server to make a Diffie-Hellman [7] key pair in the first message, we can have a protocol that can be executed in just 2 steps. This removes the need of a separate binding 'b' and β between the first and the last message, and further removes the need for the server to perform exponential calculations to verify and authenticate the client..

The protocol is shown in figure 3. Let *g* be the generator of the field over which the DH problem is intractable. The client *Alice* has a private key '*a*' and a public key $g^a$ and the server *Bob* has a private key '*y*' with a public key $g^y$. We assume that the client has a certificate $Cert_A$ from a trusted authority

where $Cert_A$ = <$ID_A$, $g^a$, SigTA($ID_A$, $g^a$ )>. Unlike the Server-Specific MAKEP algorithm in [2] we do not use a server-specific certificate and avoid scaling issues.

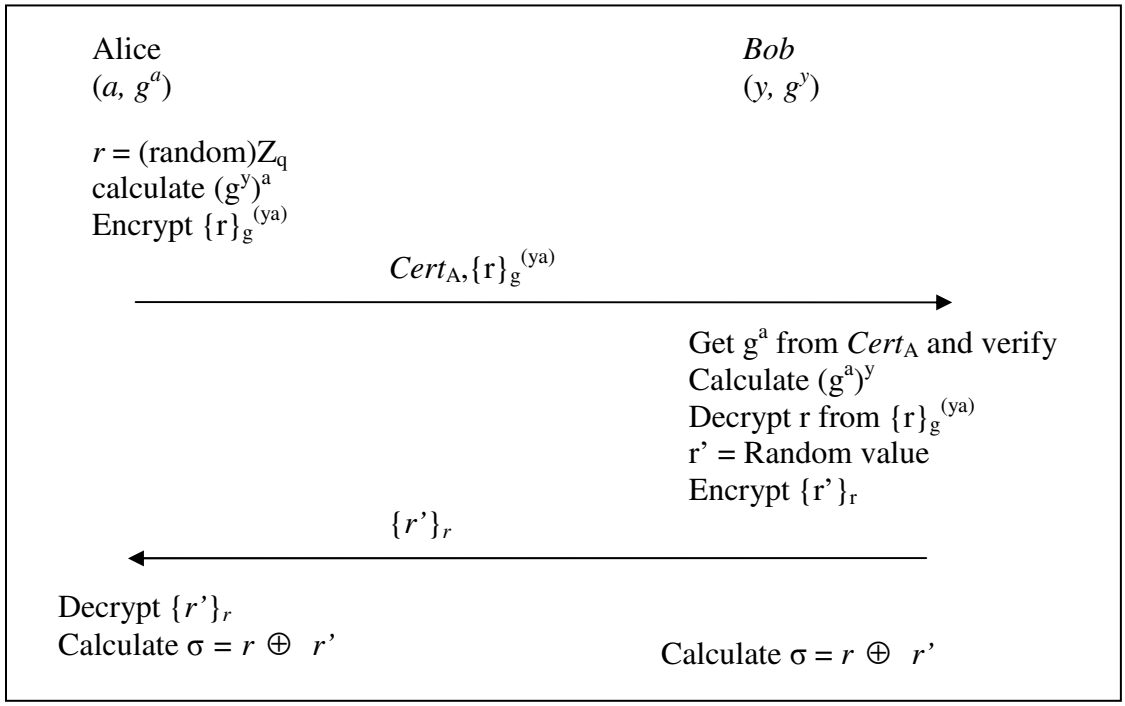


Figure 3: A new MAKEP for wireless devices

*Alice* first computes a random number *r*. She then calculates the DH key $g^{(ya)}$ using her private key *a* and Bob's public key $g^y$. Our protocol is unique in the sense that we propose that this DH key be converted to an AES key. The subsequent use of the AES symmetric key algorithm in encryption and decryption of data leads to a substantial improvement in time.

*Alice* sends her random contribution *r* encrypted under the AES key to *Bob* along with her certificate. *Bob* verifies *Alice's* public key $g^a$ and uses it to calculate the DH key $g^{(ya)}$ using his own private key *y*. He also converts the DH key to an AES key and uses it to decrypt the message and get Alice's random number *r*. Bob computes his own random number *r'* and uses *r* as an AES key to encrypt it and send it to Alice. Alice can decrypt this message since she already knows her random *r*.

σ is the new session key and H is a cryptographic hash function. By using *r* (the client random) and *r'* (the server random) we are maintaining the freshness of the session key and the $g^{(ya)}$ value is used for authentication *Alice* and *Bob* to each other, as well as preventing any other party from mounting any replay attacks or Denial of Service attacks on *Alice* or *Bob*. Since we have authenticated *Alice* to *Bob* in the first step, we don't need a third step like in [2].

Another variation of this protocol could be made using elliptic curve cryptography (ECC). A 1024 bit RSA key is comparable to only a 164 bit ECC key [8] i.e. it provides the same level of security so with the shorter key length the protocol will be much faster. In the ECC version of the protocol all the variables will now be points on an elliptic curve E. *Alice*'s private and public keys will be the field element *a* and the point *Pa* (obtained by point multiplication of the base point *P* and the secret key *a*),

while *Bob*'s keys will be *y* and *Py*. *P* is a publicly known base point. Instead of encrypting *r* with $g^{(ya)}$ we will encrypt it with (*Pya*). The rest of the protocol remains similar to the original one.

## 4. A More Secure Version of the Protocol

There is however a problem with the protocol based on trust issues. A weakness has been pointed out for the Server-Specific MAKEP protocol in [9], which is the server has control over the session key.

If we look at the server side computations before the second message is sent in figure 3, we observe that the server can decide before hand which session key (σ) will be used by calculating $r' = (r \oplus \sigma)$ and it can do this because it knows the random value of the client before the final session key is calculated.

We can easily overcome this by adding a predetermined value before the random contributions before encrypting them. By doing this, we force the participants to check if the decryption has lead to a valid result. For example, *Alice* will send $Cert_A, \{ID_A, h(r)\}_g^{(ya)}$ and *Bob* will reply with $\{ID_B, r'\}_{h(r)}$.

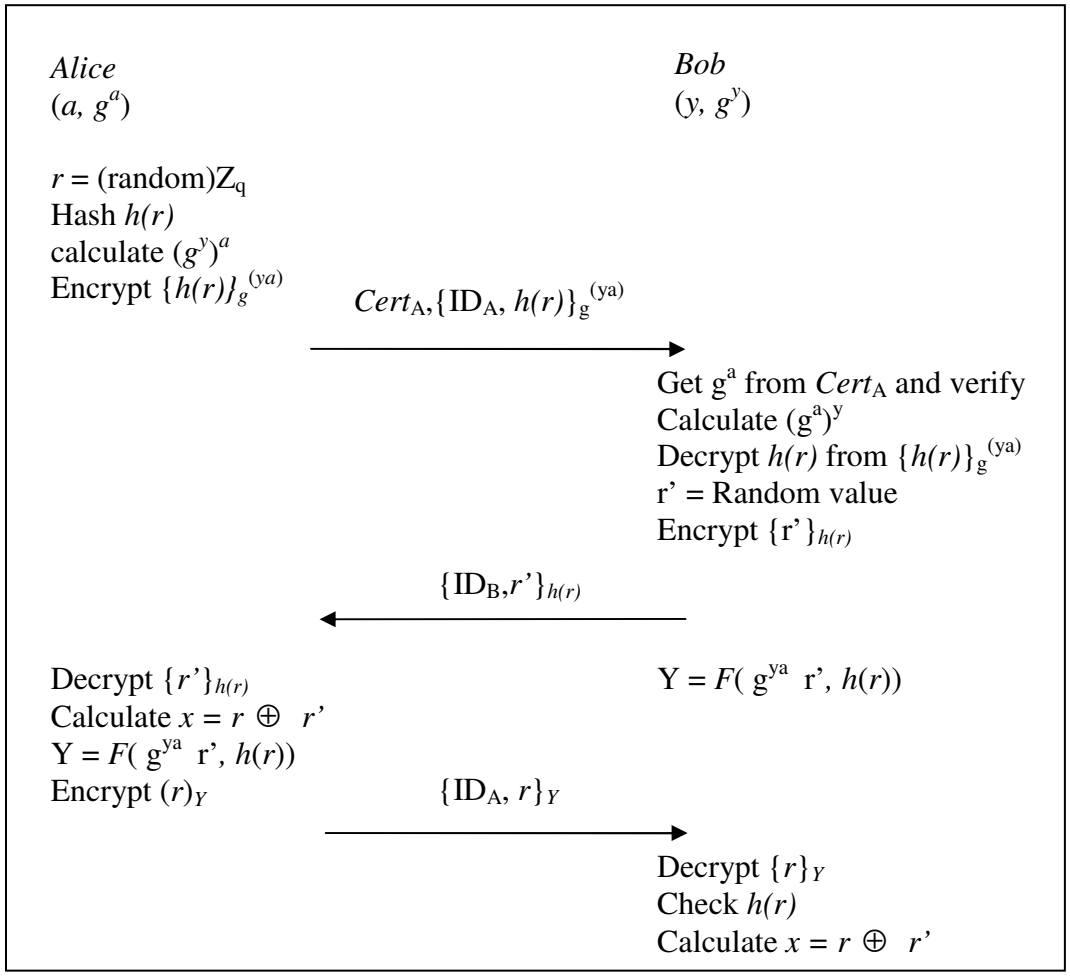


Figure 4: A more secure version

As can be seen from figure 4, an extra step has been added to the protocol. In the first step we send *h(r)*, which is the hashed value of the client random *r*. So now the server *Bob* does not know the client's random number before calculating his own random contribution *r'*. So there is no way for the server to decide the session key *x* beforehand. In the third step we send the actual client random *r*, encrypted with a symmetric key derived from the DH key $g^{(ya)}$ and the server random *r'*. Note that this will also be a symmetric key encryption and hence almost negligible in comparison to the public key operations. So the addition of the third step will not cause any difference in the speed of the protocol.

## 5. Implementation and Results

We have implemented the ECC version of our protocol and an ECC version of the Client-Server protocol. The ECC version of the Client-Server protocol has been done by replacing the exponentiation operations with point multiplication operations, and by replacing the field generator *g* with the base point *P*. By doing this we have a level playing field on which to compare both the protocols for timing efficiency using the same key sizes and the same ECC implementation.

Our implementation of ECC is based on the one described by Rosing [10]. We have used a 206 MHz 64MB RAM HP-Compaq iPAQ PDA as the client and a 1.7 GHz P4 PC as the server.

Table 1: Comparison for the total time taken

| **Field Size (Bits)** | **Our Protocol (secs)** | **Client-Server MAKEP (secs)** | **Speed comparison** |
|---|---|---|---|
| 158 | 4.81 | 15.3 | 3.18 times |
| 155 | 4.53 | 12.44 | 2.75 times |
| 134 | 3.45 | 9.3 | 2.7 times |
| 119 | 2.88 | 7.49 | 2.6 times |
| 113 | 2.79 | 7.03 | 2.51 times |
| 90 | 1.41 | 4.02 | 2.85 times |
| 65 | 1.078 | 2.06 | 1.9 times |
| 50 | 1.04 | 1.63 | 1.56 times |

As we can see from the graph in figure 5, our protocol is almost 1.5 times faster at low field sizes of around 50 bits (1.04 seconds vs 1.63 seconds) and more than 3 times faster for higher bits (at 158 bits our protocol runs at around 4.81 seconds as opposed to 15.3 seconds by the Client-Server MAKEP).

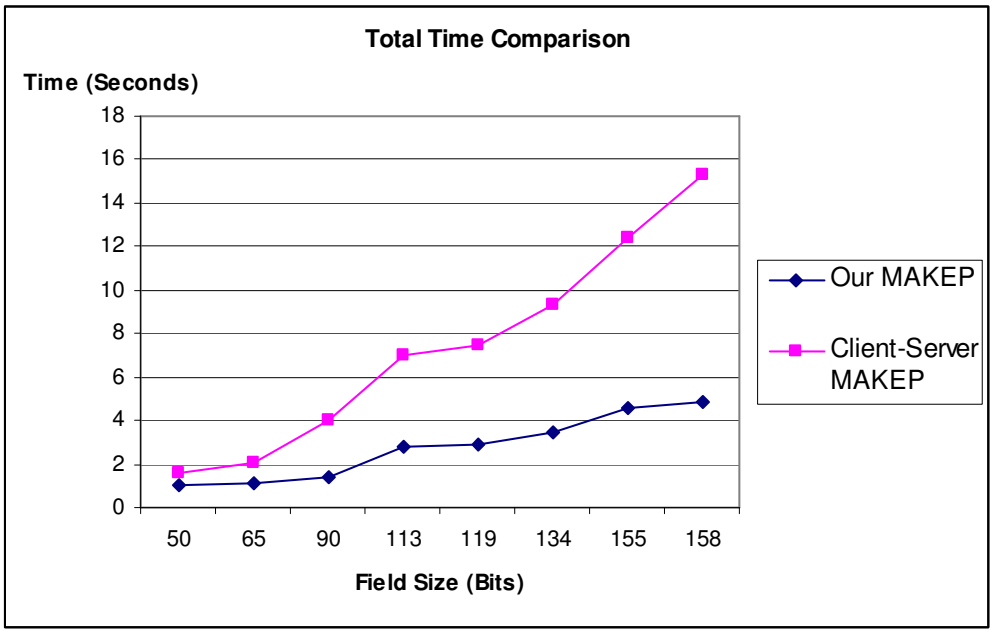


Figure 5: Graph of total time comparison

We will now analyze the two protocols by breaking them up into a pre-computation part and a run time part. The pre-computation portion is the portion of the protocol that does not need to be executed at runtime. The pre-computational time is given in the table 2.

Table 2: Runtime comparison

| **Field Size (Bits)** | **Our Protocol** | | **Client-Server MAKEP** | |
|---|---|---|---|---|
| | Pre-comp time (sec) | Run Time (sec.) | Pre-comp time (sec.) | Run Time (sec.) |
| 158 | 2.976 | 1.84 | 2.97 | 12.4 |
| 155 | 2.745 | 1.79 | 2.74 | 9.7 |
| 134 | 1.844 | 1.61 | 1.83 | 7.56 |
| 119 | 1.26 | 1.54 | 1.309 | 6.2 |
| 113 | 0.964 | 1.4 | 0.988 | 6.004 |
| 90 | 0.51 | 1.2 | 0.531 | 3.67 |
| 65 | 0.19 | 0.98 | 0.211 | 1.84 |
| 50 | 0.1 | 0.94 | 0.101 | 1.5 |

The actual time from table 1 minus the pre-computational time from table 2 gives us the run time of the protocol. Figure 6 shows these values plotted in a graph. If we compare the graphs in figures 6 and 5 we can see that our protocol is still nearly 7 times faster at the higher bit size fields and 1.5 times faster for the lower bit sizes.

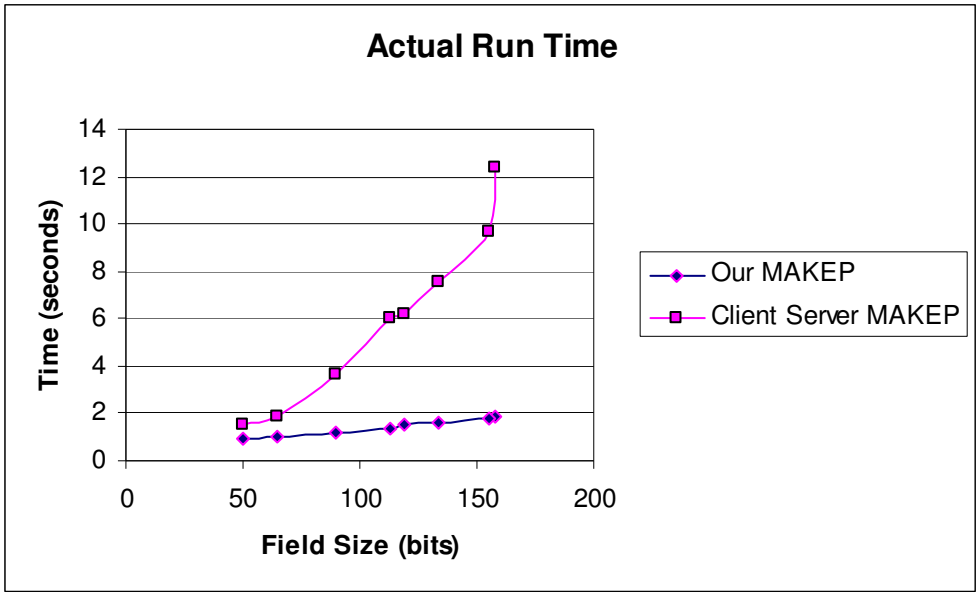


Figure 6: Actual Run time graph

We will now look at a major improvement we have made in our protocol in the first step. This is converting the DH key into an AES key and carrying out symmetric key encryption of the client random value. As opposed to this, we have used El Gamal public key encryption to encrypt the client random value in the client-server MAKEP protocol. We can see the time difference in table 3.

Table 3: Step 1 Encryption Comparison

| **Field Size (Bits)** | **Our Protocol** [AES Encrypt $r_A$ (msecs.)] | **Client-Server MAKEP** [El Gamal Encrypt $r_A$ (msecs.)] |
|---|---|---|
| 158 | 13 | 5600 |
| 155 | 12 | 5090 |
| 134 | 12 | 3500 |
| 119 | 12 | 2700 |
| 113 | 13 | 2030 |
| 90 | 12 | 1040 |
| 65 | 12 | 400 |
| 50 | 12 | 200 |

As we can see from table 3, this is a very big improvement in terms of speed.

## 6. Conclusions

We have proposed a new and efficient mutual authentication and key exchange protocol for low powered clients like PDAs in wireless environments. We have analyzed two recently proposed protocols and have implemented one of them using ECC for comparison with our protocol. We have seen that our protocol is around 3 times as fast at key sizes of around 160 bits. Furthermore, our protocol is scalable as it does not require server specific certificates. We use random contributions from both the client and the server to produces a session key that ensures forward secrecy.

## References


1. Nichols R. & Lekkas P. (2002) '*Wireless Security: Models, Threats and Solutions*' McGraw-Hill.
2. Wong D & Chan A. (2001). *'Mutual Authentication and Key exchange for low power wireless communications*', IEEE MILCOM 2001 Conference Proceedings.
3. Wong D & Chan A. (2001). *'Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices'*. ASIACRYPT 2001.
4. Schneier B., '*Applied Cryptography'*, Wiley (1996).
5. Jakobsson M. & Pointcheval D. (2001) '*Mutual Authentication for low-power mobile devices*'. Proceedings of Financial Cryptography 2001, Springer-Verlag.
6. Frankel S., Glenn R. et al. '*RFC 3602: The AES-CBC Cipher Algorithm and Its Use with IPsec'* (Retrieved September 2004 from http://rfc3602.x42.com/.
7. Diffie W. & Hellman M., *'New Directions in Cryptography'*, IEEE Transactions on Information Theory, 644-654 (1976)
8. Lenstra A., & Verheul E. (2001) *'Selecting Cryptographic Key Sizes'*, Journal of Cryptology, 14(4):255-293.
9. S.-L. Ng and C. J. Mitchell, *'Comments on mutual authentication and key exchange protocols for low power wireless communications'*, Retrieved October 2003 at http://www.isg.rhul.ac.uk/~cjm/comaak.pdf,
10. Rosing, M. (1999) *'Implementing Elliptic Curve Cryptography'*. Greenwich, CT: Manning.