R O M A N I A
MINISTRY OF NATIONAL DEFENCE
Military Technical Academy

THE 31ST INTERNATIONALLY ATTENDED SCIENTIFIC CONFERENCE
"MODERN TECHNOLOGIES IN THE XXI CENTURY"

Bucharest 03-04 November 2005

VIRTUAL PRIVATE NETWORKS POWERED BY ELLIPTIC CURVE CRYPTOGRAPHY

Eugen Petac[*]
Tudor Udrescu[**]

This article studies the impact of the implementation of Elliptic Curve Cryptography (ECC) into open-source software, and presents a user-friendly configuration tool that can be used to deploy an ECC-powered Virtual Private Network.

1. Introduction

Elliptic Curve Cryptography (ECC) is fast becoming a viable alternative to traditional public-key cryptosystems (RSA, DSA, DH). Although ECC algorithms have been available for quite some time, most of the work in this field has been theoretical in nature, with few actual implementations.

This situation has changed because of two factors. One is that processing power itself is increasing and hackers have more resources available to them than ever before. Although 1024-bit RSA keys are the most commonly used keys today, employment of 2048-bit keys is becoming more and more widespread. With the current rate of development for the IT infrastructure and machines, 10.000-bit keys are going to be needed in order to maintain the same level of security. The increase in key size, makes it impracticable to integrate traditional public-key cryptosystems into mobile/wireless devices, which are typically limited in terms of computational power, memory or network connectivity.

The alternative is to use encryption based on elliptic curves. One of the major advantages of ECC is that it offers equivalent security with RSA but uses smaller key sizes. For example, a 224-bit ECC key offers the same level of protection as a 2048-bit RSA key. This leads to increased performance in Internet communication because of faster computation times and less bandwidth being used[1].

Another reason is growing acceptance of ECC as an industry standard, which has been reflected in the work of the Internet Engineering Task Force (IETF). Elliptic Curve Cryptography can now be found in the RFCs for all the key Internet security protocols: SSL/TLS, IPSec, PKIX and S/MIME. This paper focuses on the support for ECC present in Secure Sockets Layer protocol, particularly in the OpenSSL toolkit.

Another technology that is gaining in popularity with small and medium size companies as well as universities, is VPN (Virtual Private Networks). These institutions are frequently unable, or unwilling to invest in a commercial VPN solution but still have a need for secure communication channels between different sites they posses. One solution is the deployment of open-source software. Most of the open-source VPN implementations use either the IPSec protocol (OpenSwan, FreeS/WAN, KAME) or the SSL protocol (OpenVPN).

Unfortunately, the availability of non-commercial VPN-ECC solutions still leaves much to be desired. Currently, open-source IPSec implementations like OpenSwan support only a few of the so called “Oakley Groups” as specified in RFC 2409 (groups 2 and 5 which are defined modulo primes of various length). No groups defined using elliptic curves are supported. Commercial vendors like Cisco and Nortel all support ECDH (Elliptic Curve Diffie Hellman) for key exchange in their gateways, but the financial cost associated with these products is high.

With regards to the SSL/TLS (Secure Sockets Layer/Transport Layer Security) protocol, things are much brighter for the open-source community. Sun Microsystems is promoting the adoption of ECC into the IETF standards for TLS, and has donated an ECC implementation to the OpenSSL Project, which is a general purpose cryptography library. OpenSSL now provides an implementation for the ECDH algorithm for key agreement in a TLS handshake and for the ECDSA (Elliptic Curve Digital Signature Algorithm) as an authentication mechanism. The SSL-based VPN solutions are particularly attractive as they are easier to setup and can tunnel traffic over a single UDP or TCP port making the configuration of the firewall a trivial matter. OpenVPN can use all of the encryption, authentication, and certification features of the OpenSSL library to protect private network traffic as it transits the Internet [2]. Thus, with the addition of ECC capabilities to OpenSSL, this can be used in conjunction with OpenVPN to provide a powerful and low-cost VPN solution which can accommodate a wide range of configurations, including remote access, site-to-site VPNs and WiFi security.

As an increasing number of users grow more sensitive to security and privacy concerns, closer integration of different open-source technologies like OpenSSL and OpenVPN is needed to provide easy access to efficient cryptographic techniques like ECC. To facilitate integration and deployment of these technologies over the industry and the academic world, we have developed an easy-to-use tool, that enables an user with limited skills to create an ECC-enabled VPN.

2. SSL Overview

Secure Sockets Layer [2] is the dominant security protocol on the Internet today. Like most Internet security protocols, SSL employs a public-key cryptosystem to derive symmetric-keys and then uses fast symmetric-key algorithms to offer encryption, source authentication and integrity protection for data exchanged over insecure, public networks. One of the major strengths of SSL, is its ability to use a variety of different cryptographic algorithms for key agreement, symmetric encryption and hashing. Although, we can use any combination of cryptographic algorithms, it is strongly recommended we use so called *cipher suites*, which are well tested combinations. For example, the cipher suite RSA-RC4-SHA contains RSA as the key exchange mechanism, RC4 for symmetric encryption and SHA as the hash function. Due to significant advances in cryptanalysis and computing power available to potential intruders, in the future, both symmetric and public key-sizes must grow from current levels in order to ensure the same security as today. Because public-key cryptographic operations are the most computationally expensive part of an SSL transaction, a replacement for RSA must be found. As shown in Table 1 [3], Elliptic Curve Cryptography offers a significant strength per bit when compared to RSA. The table contains the equivalent key sizes for symmetric algorithms, ECC and RSA, as well as the key size ratio between traditional public-cryptosystems and elliptic curve based cryptosystems.

Symmetric	ECC	RSA	Key size ratio
56	112	512	5:1
80	163	1024	6:1
112	224	2048	9:1
128	256	3072	12:1
192	384	7680	20:1
256	512	15360	30:1

Table 1 : Computationally equivalent key sizes

Due to the use of smaller keys for equivalent security levels, ECC is especially suited for implementation in mobile/wireless devices, where power, bandwidth and CPU are constrained. Applications like online banking or secure wireless communications, that make use of digital signatures and authentication, also benefit from the reduction in key sizes.

3. ECC Overview

The application of elliptic curves in cryptography was first proposed in 1985 independently by Neal Koblitz [4] from the University of Washington, and Victor Miller [5] from IBM. Many cryptosystems require the use of algebraic groups. Elliptic curves can be used to form elliptic curve groups, where the elements of the group are points on an elliptic curve. The introduction of more stringent properties for the elements of the group, such as limiting the number of points on such a curve, creates an underlying field for the elliptic curve group. In practice, Elliptic Curve Cryptosystems operate over F_p fields (where p is a prime) and $F_2{}^m$ fields (a binary representation with 2^m elements). The core of elliptic curve arithmetic is the operation called *scalar point multiplication*, which computes Q = *k*P, where Q is a point on the curve obtained by multiplying *k* times another point P, also on the curve. Scalar multiplication is performed by a combination of point-additions (defined as the operation of adding two distinct points together) and point-doublings (which add two copies of a point together) [6]. Both point-additions and point doublings can be implemented very efficiently both in software and in hardware.

At the foundation of every cryptosystem is a hard mathematical problem that is computationally unfeasible to solve. For instance, RSA relies on the problem of integer factorization, while Diffie-Hellman uses the discrete logarithm problem. Unlike these cryptosystems which operate over integer fields, the Elliptic Curve Cryptosystems operate over points on an elliptic curve and rely upon the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP). This problem states that given two points on the elliptic curve, P and Q =*k*P, it is computationally unfeasible to find *k*. An Elliptic Curve Cryptosystem is made up of domain parameters for the elliptic curve. Besides the curve equation, one of the most important elliptic curve parameters is the *base point* G, which is fixed for each curve. By multiplying the curve's base point G with *k*, we obtain the point Q which is the public key. The private key is represented by *k*, which is a large random integer.

A variety of attacks against Elliptic Curve Cryptosystems like *Pollard rho* or *Pohlig-Hellman* [7], have been devised. Their efficiency varies depending on the choice of the elliptic curves. To guarantee a high level of security, only curves recommended by standards organizations like NIST should be used.

Elliptic Curve Diffie Hellman (ECDH) and Elliptic Curve Digital Siganture Algorithm(ECDSA) are the elliptic curve counterparts of the Diffie-Hellman key exchange and Digital Signature Algorithm, respectively. Figure 1 provides a description of the ECDH key agreement between two users, Alice and Bob.

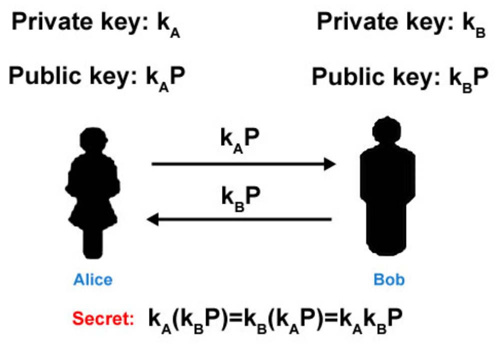


Fig 1 : ECDH key agreement

The two users generate their respective private keys, k_A and k_B and corresponding public keys $Q_A= k_AG$ and $Q_B= k_BG$, where G is the base point of the elliptic curve they have agreed to use. The parties then exchange their public keys, and compute the shared secret by multiplying their own private key with the other's public key.

4. Public-key cryptography in SSL

The SSL/TLS protocol is composed of several specialized protocols, with the Handshake protocol and the Record protocol being the most important. SSL Handshake is responsible for the establishment of a SSL session between the client and the server, allowing the two parties to negotiate a common cipher suite, authenticate each other, and derive a shared master secret using public-key cryptographic algorithms. The SSL Record protocol derives symmetric keys from the master secret and uses fast symmetric encryption algorithms for the bulk encryption and authentication of application data.

The most computationally expensive part of an SSL transaction, is represented by the public-key cryptographic operations. Once a master secret has been derived, it can be re-used, resulting in a abbreviated handshake that does not contain any public-key cryptographic operations. A client and server must always perform a full-handshake on their first interaction. Figure 2 shows the operation of an ECC-based SSL handshake, as specified in [3].

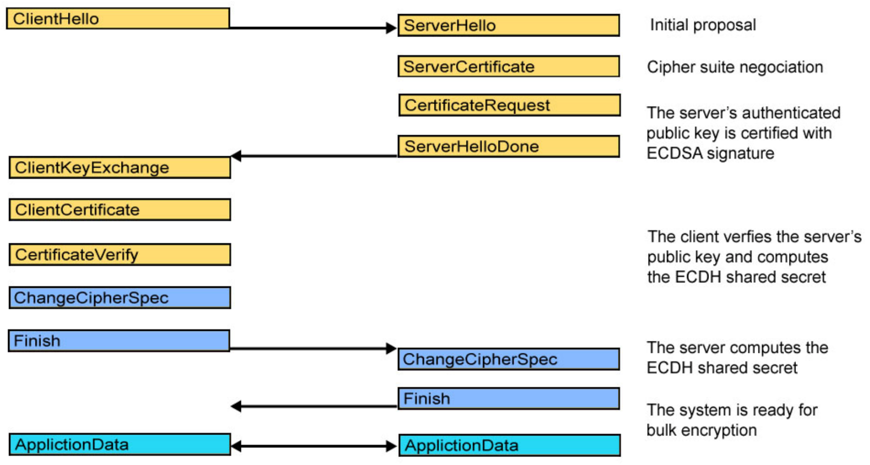


Fig. 2 : ECC-based SSL Handshake

There are two variants of the ECDH-ECDSA handshake. The first one doesn't use client authentication. In this case, the client performs an ECDSA verification to check the server's ECDSA certificate and then an ECDH operation using its private ECDH key and the server's public ECDH key to compute the shared *pre-master*. The server only performs an ECDH operation to arrive at the same secret [7].

When client authentication is used, the actual messages being exchanged are dependent on the type of authentication requested by the server and the kind of certificate the client posses. If the client uses an ECDH certificate, both sides perform an ECDSA verification on the other's certificate followed by an ECDH operation to compute the *pre-master* secret. If the client uses an ECDSA certificate, the operations required on the two sides are asymmetric. The client performs an ECDSA verification of the server's certificate, an ECDH operation to compute the *pre-master* secret and an ECDSA signature to generate the *CertificateVerify* message. The server performs an ECDH operation to compute the *pre-master* secret and two ECDSA verifications (one to verify the client's certificate and another to verify the *CertificateVerify* message) [7].

5. Performance evaluation and analysis

In order to evaluate the performance of an ECC-powered VPN, we used the built-in speed program from the OpenSSL toolkit. Table 2 shows the measured performance for RSA, ECDH and ECDSA public-key operations on a platform equipped with a 1.5 Ghz AMD Athlon processor. The top row shows the time needed for one primitive public-key operation for 1024-bit RSA and 163-bit ECC, whereas the bottom row displays the time for 2048-bit RSA and 192-bit ECC [7]. The values listed are in milliseconds.

$RSA_{encrypt,verify}$	$RSA_{decrypt,sign}$	$ECDSA_{verify}$	$ECDSA_{sign}$	$ECDH_{op}$
0.2	4.7	3.0	0.6	2.6
0.7	24.6	6.2	2.9	3.0

Table 2. Measured performance of public-key operations (in milliseconds)

Next, we used these values to compare the performance of RSA and ECC SSL-handshakes. The metric used was the Handshake Crypto Latency, defined as the total amount of time spent on performing cryptographic operations on the client and on the server [7]. Figure 3 illustrates the impact of using higher key sizes on an SSL handshake. It is clear that the performance advantage of ECC over RSA increases at higher key sizes. While the latency for 1024-bit RSA is better than that for 163-bit ECC, in the case of 2048-bit RSA, things are different, and ECC is up to 3 times faster.

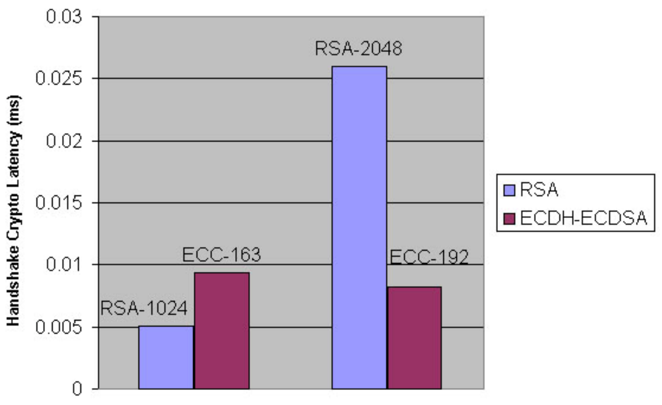
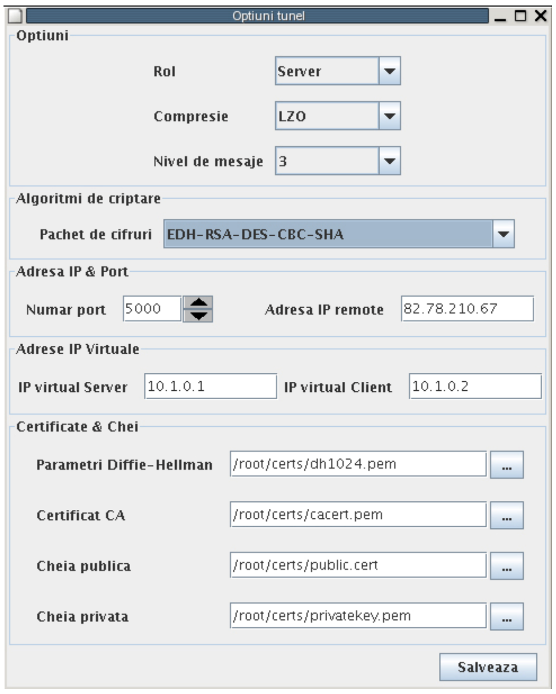


Fig.3 : Impact of using higher key sizes

6. Implementation of an ECC-powered VPN

In order to implement an open-source ECC-powered VPN, we developed an application stack and an installation methodology. The application stack consisted of an ECC-enabled version of the OpenSSL cryptographic library with support for ECDH and ECDSA, the latest version of OpenVPN, the LZO data compression library, and a configuration utility specially developed for the project. The installation procedure was successfully tested on several Linux distributions (RedHat 9, Ubuntu 5.04, Fedora Core 2) and should, with minor modifications, work on other distributions as well. Because the configuration utility was developed using Java, a version of the Java 2 platform for Linux greater than 1.4.2 must also be installed on the target machine. We tested the operation of the VPN first in a laboratory environment inside Ovidius University of Constanta, and then by creating a VPN between the University and an offsite location. We also simulated multiple clients that were accessing the internal university network using the VPN gateway we had set up, thus demonstrating the feasibility of deploying this open-source solution to give faculty staff and students access to internal resources. During tests, the encrypted tunnel proved stable.

Because the open-source solution is made up of several independent components, that are developed separately, their integration and successful configuration requires considerable skills. To simplify the process of establishing an encrypted tunnel, we have developed a configuration utility for VPNs based on the SSL protocol.



Optiuni
Rol: Server
Compresie: LZO
Nivel de mesaje: 3

Algoritmi de criptare
Pachet de cifruri: EDH-RSA-DES-CBC-SHA

Adresa IP & Port
Numar port: 5000
Adresa IP remote: 82.78.210.67

Adrese IP Virtuale
IP virtual Server: 10.1.0.1
IP virtual Client: 10.1.0.2

Certificate & Chei
Parametri Diffie-Hellman: /root/certs/dh1024.pem
Certificat CA: /root/certs/cacert.pem
Cheia publica: /root/certs/public.cert
Cheia privata: /root/certs/privatekey.pem

Fig. 4 : Configuration utility screenshot

7. Conclusions

The performance analysis and the tests carried out, suggest that the use of ECC cipher suites can offer significant performance benefits to VPNs especially as security needs increase. We have completed implementing an application stack that can be used to deploy an open-source VPN solution and we are in the process of setting up a fully integrated VPN package that would simplify further the installation and configuration of virtual private network. As soon as support for ECC is available in other open-source VPN software, we intend to investigate the performance and impact of using ECC-based cipher suites in these protocols and compare the results with those obtained for the SSL protocol.

References

[1]. V. Gupta, D. Stebila, S. Fung, S. Chang, N. Gura, H. Eberle. “Speeding up Secure Web Transactions using Elliptic Curve Cryptography“. *11th Ann. Symp. on Network and Distributed System Security* — NDSS 2004, Internet Society, February 2004

[2]. A. Frier, P. Karlton, P. Kocher, “The SSL Protocol Version 3.0”, http://home.netscape.com/eng/ssl3

[3]. V. Gupta, S. Blake-Wilson, B. Moeller, C. Hawk, “ECC Cipher Suites for TLS”, IETF internet draft <draft-ietf- tls-ecc-05.txt>, Jan. 2004.

[4]. N. Koblitz, “Elliptic curve cryptosystems”, *Mathematics of Computation*, 48:203-209, 1987.

[5]. V. Miller, “Uses of elliptic curves in cryptography”, Lectures Notes in Computer Science 218: Advances in Cryptology – CRYPTO `85, pages 417-426, Springer-Verlag, Berlin, 1986

[6]. Elliptic Curve Cryptography, Certicom Research, www.certicom.com

[7]. Performance Analysis of Elliptic Curve Cryptography for SSL, *ACM Workshop on Wireless Security (WiSe), Mobicom 2002*, Atlanta, Sept. 2002

[8]. Code&Cypher Vol. 1, no. 4 , www.certicom.com/codeandcipher, Certicom’s Bulletin of Security and Cryptography, 2004.

[9]. OpenSSL Project, http://www.openssl.org

[10]. OpenVPN, http://openvpn.net/

[11]. Next Generation Crypto, http://research.sun.com/projects/crypto

* Associate Professor Engineer, “Ovidius” University Constanta, Faculty of Mathematics and Informatics, 124 Av. Mamaia, e-mail: epetac@univ-ovidius.ro

** Research Assistant, “Ovidius” University Constanta, Faculty of Mathematics and Informatics, 124 Av. Mamaia, e-mail: tudrescu@univ-ovidius.ro