www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 11 November, 2013 Page No. 3238-3244

Cryptography Implementation in IP Datagram

C.Sajeev, C.Suyambulingom
Research Scholar , Sathyabama University
csajeev@live.com, csajeevpkm@gmail.com
,
Professor (Rtd.), Tamilnadu Agricultural University , Coimbatore

Abstract—The IP packets prepared to sent across the network have a digital signature and public key attached to it which allow to check on each hop along the route to verify the authenticity of packets. For this we use ECC (Elliptic Curve Cryptography). In this paper, we present a software solution of cryptography for PLA (Packet Level Authentication) using the combination of koblitz curves to increase throughput and implicit certificates to decrease storage and computation overhead. A software is developed on open SSL libraries and extending the Open SSL API to support not only fast ECC using Koblitz curves, but implicit certificates and fast signature verifications using implicit certificates as well. Software implementation results of these API extensions are provided, yielding significant speedup of elliptic curve operations.

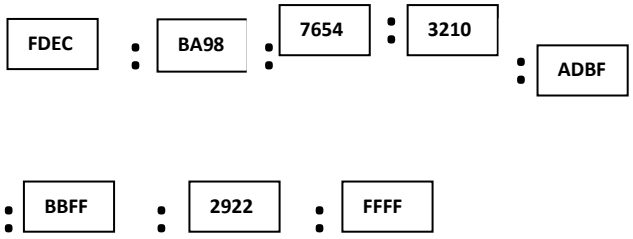
Key Words—Elliptic curve cryptography, Koblitz curves, digital signatures, implicit certificates, datagram.

I. INTRODUCTION

A large, shared resource will inevitably be exploited by its users. This idea is timeless and has been documented as far back as 350 BC by Aristotle. For that which is common to the greatest number has the least care bestowed upon it. Every one thinks chiefly of his own, hardly at all of the common interest; and only when he is himself concerned as an individual.

The same is true of the Internet. Various types of attacks are widespread. New and more efficient countermeasures are constantly required to protect against such attacks. Packet Level Authentication (PLA) is one such countermeasure which provides protection at the network infrastructure level.

PLA modifies IPv6 packets by adding a so-called PLA header to each packet.



Sequence numbers and timestamps are added to ensure timeliness and uniqueness. Certificate, digital signature, and trusted third party ID fields are added. Users are authenticated by a trusted third party, then include this information along with the signature on the packet in the corresponding fields. This allows every hop along the route to verify the authenticity of the packet. This does not require previous communication between the sender and the receiver. The authenticity can only be verified once the packet has reached the final destination—thus IPSec is not a good countermeasure to prevent distributed denial-of-service. To implement PLA, signatures are quickly verified to increase throughput, but at the same time be compact enough as to not cause excess overhead in the packet. In this paper, we present a software solution for PLA cryptography. The combination of Koblitz curves and implicit certificates yields an interesting cryptographic setting, ripe with methods for speeding up the cryptographic operations. It provides Open SSL API extensions for high-speed Koblitz curve support and implicit certificates. Results from a software implementation of these Open SSL API extensions are provided. The result is high-speed, compact, and secure elliptic curve cryptography, for use not only with PLA, but for the Open SSL community and elliptic curve cryptography in general.

II. PRACTICES MADE ON PLA

PLA needs cryptography parameters and have chosen in such a way as to make signature verifications fast, while at the same time minimizing the packet overhead. This is done using a combination of Koblitz curves and implicit certificates. This results in fast and compact cryptography; a detailed description follows.

A. Koblitz Curves

Koblitz curves are elliptic curves over F_2 defined by the equation

$$E_a(F_2m) : y^2 + xy = x^3 + ax^2 + 1 \text{ where } a \in \{0, 1\}. \quad (1)$$

The set of all (x, y) solutions over F_2m along with the identity element O, or point-at-infinity, form an abelian group with point addition (doubling) that can be used for cryptographic operations [1], [2]. These curves are of particular interest in high-speed cryptography as they admit the efficient endomor-phism $\varphi : E_a(F_2m) \rightarrow E_a(F_2m)$ defined by

$$\varphi(x, y) \rightarrow (x^2, y^2), \varphi(O) \rightarrow O \quad (2)$$

called the Frobenius endomorphism. Squaring is a linear oper-ation in F_2m so this endomorphism can be applied extremely efficiently. It can be shown from the point addition formula that φ satisfies the characteristic polynomial

$$P(T) = T^2 - \mu T + 2 \text{ where } \mu = (-1)^{1-a}$$

and denoting τ as a complex root of (3), applying φ carries out complex multiplication by the complex number $\tau = (\mu + \sqrt{-7})/2$. Integers are then considered as members of $Z[\tau]$ and expressed as the sum (and difference) of powers of τ , a base-τ expansion:

$$K = \sum_{i=0} c_i T^i \quad (4)$$

where ci are the coefficients. Scalar multiplication, the main operation in elliptic curve cryptography, is then accomplished using no point doublings, only applications of τ and point additions:

$$kP = \sum_{i=0} c_i \varnothing^i (p) \quad (5)$$

For PLA, currently the standardized Koblitz curve K-163 is used. With K-163, m = 163, a = 1 and r ≈ 2163. For comparison, this provides an equivalent level of 80 bits of symmetric cryptography security or 1024 bits compared to DSA/RSA. The key size requirement for elliptic curves grows much slower than those of DSA/RSA.

Koblitz curves were chosen for speed. Due to point doublings being replaced by applications of the Frobenius, other methods for general curves (or hyperelliptic curves) simply cannot compete with the reduced number of operations.

B. Authentication through ECDigital Signatures

The Elliptic Curve Digital Signature Algorithm(ECDSA) is a widely used signature scheme that uses elliptic curves. ECDSA has three phases, key generation, signature generation,and signature verification.

ECDSA Key Generation:
An entity A's key pair is associated with a particular set of EC domain parameters D= (q, FR, a, b, G, n, h). E is an elliptic curve defined over Fq,and P is a point of prime order n in E(Fq), q is a prime. Each entity A does the following:
1. Select a random integer d in the interval [1, n- 1].
2. Compute Q = dP.
3. A's public key is Q, A's private key is d.

ECDSA Signature Generation:.
To sign a message m, an entity A with domain parameters D= (q,FR, a, b, G, n, h) does the following:
1. Select a random or pseudorandom integer k in the interval [1,n-1].
2. Compute kP =x1, y1 and r= x1 mod n (where x1 is regarded as an integer between 0 and q-1). If r= 0 then go back to step 1.
3. Compute k-1mod n.
4. Compute s= k-1 {h (m)+ dr} mod n, where h is the Secure Hash Algorithm (SHA-1). If s = 0, then go back to step 1.5. The signature for the message m is the pair of integers (r, s).

ECDSA Signature Verification:
To verify A's signature (r, s) on m, B obtains an authenticated copy of A's domain parameters D = (q, FR, a, b, G, n, h) and public key Q and do the following
1. Verify that r and s are integers in the interval [1, n-1].
2. Compute w = s-1mod n and h (m)
3. Compute u1 = h(m)w mod n and u2 = rw mod n.
4. Compute u1P + u2Q =(x0, y0) and v= x0 mod n.
5. Accept the signature if and only if v = r

1

C. Implicit Certificates

Using traditional certificate-based PKI, users must have their public keys authenticated by a Trusted Third Party (TTP) to guarantee their authenticity. The TTP constructs a certificate containing many fields such as the TTP signature, validity period, etc.—the TTP then signs this certificate and appends the signature to it. When a user wants to verify a signature on a message, they should first verify the authenticity of the certificate, thus requiring even more signature verifications. In such a way a user can make their way up a chain of trust to verify if a certificate is valid or invalid.

Implicit certificates (self-certified keys) are an efficient alternative to this approach. Instead of verifying certificates and public keys using an explicit TTP signature, the public key is extracted directly from TTP's signature on the user's identity using a cryptographic operation. This reduces the storage and computational requirements.

While the extracted public key cannot be explicitly verified, resulting signatures will not verify unless the extracted key is authentic; that is, the authenticity is said to be implicit. If the message signature fails to verify, it is unknown whether the user's signature on the message is

invalid or the extracted public key is invalid (or both)—but this distinction makes little difference in practical applications.

To better define the notion of trust with respect to implicit certificates, Girault [12] introduced three distinct trust levels associated with self-certified keys.

- Trust Level 1. TTP knows the user's private key and can therefore impersonate the user without being detected.
- Trust Level 2. TTP does not know the user's private key, but can still impersonate the user without being detected.
- Trust Level 3. TTP does not know the user's private key, but can impersonate the user. However, such impersonationcan be detected.

Here, detected means that if TTP tries to impersonate a user, the user can prove it—for example, providing two different signatures from TTP on the same identity.

Trust Level 1 is inadequate for many reasons, one being that it usually requires a secure key escrow. Reaching Trust Level 3 is generally the goal; for good reasons, users are often not comfortable sharing their private keys with TTP. Consider the following scenario. An ISP (the user's TTP) charges based on bandwidth usage. Each packet is digitally signed by the user as with PLA, providing assurance that the ISP is billing in an honest manner. If the ISP can impersonate the user in an undetectable manner, the ISP can generate false traffic from the user to increase the charges. Trust Levels 1 and 2 are therefore inadequate. This is just one example of why Trust Level 3 is desirable.

1) An Implicit Certification Scheme: An implicit certification scheme based on the Nyberg-Rueppel signature scheme is given in. We presented the modified version from which omits the proof-of-knowledge step. It reaches Trust Level 3 by blinding TTP from the user's private key.
2) Setup. Elliptic curve E is chosen with base point generator G of prime order r where r | #E.

Keygen. The following protocol is used to generate a key pair on user Alice's identity IDA.

TTP ← Alice: kAG
TTP: (rA, bA) = COMPRESS(kAG + kTG)
rA = rA + H(IDA)
sA = kT − rAsT (mod r)
Alice ← TTP: (rA, bA, sA)

(6)

Alice's private key is sA = kA + sA (mod r) .

Extract. To extract Alice's public key WA = sAG on identity IDA given public values (rA, bA), Bob calculates

$$WA = \text{DECOMPRESS}(rA - H(IDA), bA) - rAWT \quad (7)$$

The extracted public key is correct (WA = sAG):

$$\begin{aligned} WA &= \text{DECOMPRESS}(rA - H(IDA), bA) - rAWT \\ &= \text{DECOMPRESS}(rA + H(IDA) - H(IDA), bA) - rAWT \\ &= kAG + kTG - rAsTG = (kA + kT - rAsT)G \\ &= (kA + sA)G = sAG \end{aligned}$$

2) Attempting Impersonation Attacks: In contrast to the key issuing protocol in , no proof-of-knowledge is performed by Alice in (6); TTP has no guarantee that Alice knows the discrete log of kAG. It is possible that a malicious user Malice is attempting to obtain a valid signature from TTP on Alice's identity—an impersonation attack. To succeed in such an attack, Malice must choose some difference d 2 Zr such that the following equation holds:

$$[(kA + d)G + kTG]x + H(IDM) = [kAG + kTG]x + H(IDA).$$

Letting k denote kA + kT (k is as random as kT is), this becomes

$$[(k + d)G]x - [kG]x = H(IDA) - H(IDM). \quad (8)$$

In general, the ability to find such a difference is linked to the differential uniformity of the projections involved. It was shown in [19] that the projection of an elliptic curve point to its x-coordinate is 4-uniform and thus the probability of successfully carrying out an impersonation attack is negligible in the group order r. In this case, the proof-of-knowledge can therefore be omitted, saving one roundtrip communication.

D. Implicit Certificates verification

In [21] it was shown how to perform self-certified public key extraction and signature verification simultaneously using the Nyberg-Rueppel signature scheme. Indeed, the same can be accomplished here with the ECDSA scheme and the implicit certification scheme above. In ECDSA signature verification shown in Sec. II-B, the value uG + vW is computed. The public key W would first be extracted using (7). Combining these two calculations, we have

$$uG + v(\text{DECOMPRESS}(rA - H(IDA), bA) - rAWT)$$

and denoting the resulting point from decompression as D,

$$uG + v(D - rAWT) = uG + vD - vrAWT \quad (9)$$

Therefore we first perform the point decompression, then calculate the value $-vr_A \bmod r$ and perform the three needed scalar multiplications simultaneously similarly as with Shamir's trick. The online precomputation requirement in-creases to 10 points.

E. Implications for PLA

A comparison of PLA with traditional PKI and with implicit certificates is given in Table I, where r is the group order, m the field size, and ESM the elliptic scalar multiplication operation. The certificate therein is a minimal one, containing only a TTP signature on the client's public key—in practice more information is needed (validity period, etc.), but only the cryptographic storage requirements are measured here.

One could argue that to save computation time, in traditional certificate-based PKI the certificate need only be verified once, then a hash stored (the same can be said of implicit certificates and extracting a client's public key). However, this requires extra storage and time and as the number of clients trusted third parties grows, this is not convenient. Additionally, the 4 scalar multiplications required for PLA with certificate-based PKI can be reduced to 2 simultaneous scalar multiplications, and similarly with implicit certificates from 3 scalar multi-plications to 1 simultaneous scalar multiplication—hence one could argue the exact opposite when using implicit certificates: such storage and hashing will in fact not significantly reduce the computation time, but simply increase the implementation complexity.

III. SOFTWARE IMPLEMENTATION

Due to its widespread use, OpenSSL is used as a basis for the software implementation. We first discuss the existing OpenSSL ECC functionality, then present extensions to the OpenSSL API to support Koblitz curves and implicit certificates. Finally, results of the implementation of these OpenSSL API extensions are provided.

A. ECC in OpenSSL

Scalar multiplication in OpenSSL is provided with the function EC_POINTs_mul that takes, amongst many argu-ments, a scalar to multiply by the generator (optional) and an array of scalars and points. The function results in one point, the sum of all these scalar multiplications. In the case of binary curves, this function calls ec_GF2m_simple_mul. which scalar multiplication method is then used depends on the number of scalars passed and if one of the points is the generator (thus the precomputation is done offline and as a result the scalar multiplication is faster). If there is only one point and it is the generator, or there are 3 or more points, the function ec_wNAF_mul is called, a standard implementation of NAF_w scalar multiplication (us-ing interleaving when more than one point is passed) which, in the case of binary curves, uses affine coordinates— the functions (via wrappers) ec_GF2m_simple_add and ec_GF2m_simple_dbl for point addition and doubling, respectively, executed at a cost of 1S + 2M + 1I where S, M, I are respectively field squarings, multiplications, and inversions. Width-w NAFs are generated using the function compute_wNAF (note that OpenSSL offsets the widths by one—thus what would usually be referred to as $NAF=NAF_2$, no two adjacent coefficients are both non-zero, would have w = 1 in OpenSSL). If there is only one point passed and it is not the generator, or there are exactly two points passed, the function ec_GF2m_montgomery_point_multiply is then iterated to obtain the resulting point. This function is a straightforward implementation of Montgomery's ladder using projective coordinates from that eliminates the need for precomputation.

In OpenSSL, public and private key pairs are generated using the EC_KEY_generate_key function. ECDSA sig-natures are created using ECDSA_do_sign and verifications done using ECDSA_do_verify.

To summarize, for PLA and Koblitz curve K-163 using a mostly stock OpenSSL this means for generating ECDSA signatures, the standard NAF_4 scalar multiplication with affine coordinates is used with offline precomputation. For extracting the public key from the implicit certificate, Montgomery's ladder is used with projective coordinates. For verifying ECDSA signatures, Montgomery's ladder is iteratively used with projective coordinates. For simultaneous key extraction and signature verification (requiring some modifications to the stock OpenSSL to support implicit certificates as described below), the interleaving method with NAF_4 is used for scalar multiplication with affine coordinates; the precomputation for the generator is provided offline, but the precomputation for the remaining two points is done online.

B. OpenSSL Extension

Adding support in OpenSSL for the implicit certificate scheme (6) previously presented is fairly straightforward. We use the naming convention ICNR for "Implicit Certificate, Nyberg-Rueppel". For the key issuing protocol, the function ICNR_blind generates the blinded value for users, basically just a wrapper for key generation; the result can then be passed to ICNR_ttp that produces a TTP signature on the identity and certificate. Finally, ICNR_finalkey computes the user's final private key given the blinded value and the result from the TTP. To extract (7) the keys, ICNR_extract is used. The simultaneous extraction and signature verification is provided by ICNR_do_verify—optionally, keys can first be extracted using ICNR_extract then signatures verified using the standard

ECDSA_do_verify as usual.

The first step to speed up scalar multiplication using binary curves in OpenSSL is to provide projective coordinates, which trade the (relatively) expensive field division in point addition and doubling present when using affine coordinates for a number of field multiplications and squarings. Lopez´-Dahab (LD) projective coordinates were implemented, resulting in a point addition cost of 8M+5S using ec_GF2m_ld_add and point doubling 4M + 5S using ec_GF2m_ld_dbl. OpenSSL does not have any special methods for Koblitz curves—they are treated as normal binary curves. We extend OpenSSL to support Koblitz curve-specific functions.

Method	Storage	Time (ms)
Unmodified OpenSSL		
Sign	3/0	4.104
Extract	0	4.214
Verify	0	8.289
Extract+Verify	3/6	25.461
Modified OpenSSL		
Sign	0/2	2.374
Extract	0/2	2.472
Verify	0/2	4.376
Extract+Verify	0/10	5.966

We implemented a function compute_koblitz_reduce that performs reduction of scalars as described in Sec. II it is important call this function to shorten the of scalars before scalar multiplication, otherwise the op-eration will take roughly twice as long. An analogous function compute_wNAF_tau is implemented that gen-erates width−w τ NAFs as described in. The function ec_GF2m_koblitz_tau applies the Frobenius (2)—in the case of projective coordinates, a minor cost of 3S for precomputation to be done in projective coordinates, then all at once normalized to affine coordinates. However, the resulting difference in timings was negligible and hence the results are not included.

TABLE I
PLA STORAGE AND COMPUTATION REQUIREMENTS COMPARED

Certificate-Based PKI	PLA	Implicit Certificates	PLA
signature (2r)	326	signature (2r)	326
public key (m + 1)	164	self-certified public key (m + 1)	164
TTP signature on public key (2r)	326	-	0
verify public key	2 ESMs	extract public key	1 ESM
verify signature	2 ESMs	verify signature	2 ESMs
Packet Total (bits)	816		490
Computation (ESMs, Sign/Verify)	1/4		1/3

For scalar multiplication using Koblitz curves, we im-plement the function ec_GF2m_koblitz_mul, somewhat analogous to ec_GF2m_simple_mul. When only one point is passed to the function, the the sliding window method on the τ NAF_2 is used, with a window width of w = 3; for two or three points, a method using Shamir's trick with τ -adic scalar recoding specific to Koblitz curves. For two scalars, two precomputation points must be computed online and this method produces a joint weight (0.5) with the same average length and density as that of τ JSF. For three scalars, 10 precomputation points must be computed online and the average joint weight is 0.5897. We explicitly provide the three-term scalar multiplication algorithm with recoding that can be used for simultaneous key extraction and signature verification in Fig. 2. The precomputed points must be in affine coordinates. In the precomputation phase, we also implemented a version of Montgomery's trick for simultaneous inversion. This would allow the point additions

C. Implementation Results

The implementation was done on an AMD Athlon Thun-derbird 1.0GHz 32-bit processor with 1GB of RAM running Debian Linux and OpenSSL v0.9.8g. The compiler used was GCC v4.1.2 using compiling switches -march=athlon-tbird -O3 -pipe -fforce-addr -fomit-frame-

pointer -funroll-loops. The timing results are given in Table II. The "Unmodified OpenSSL" section is the stock OpenSSL with only minor mod-ifications to support implicit certificates and simultaneous key extraction and signature verification ("Extract+Verify"). The "Modified OpenSSL" section includes the entire implementation.

Input: `-bit expansions a, b, c in _NAF2, points P, Q,R 2 E1(F2m)
Output: aP + bQ + cR
Precompute xP + yQ + zR 8 x, y, z 2 {−1, 0, 1}
S ←∞ 1, i ← l − 1
while i> 0 do
D ← {a, b, c},C ← 1
foreach k € D do
if ki = 0 then D← D \ {k}
else if ki + ki−2 = ±2 then C max(2,C)
else if ki + ki−3 = ±2 then C max(3,C)
else D ←Ø, C ← 1
end
for j ← 1 to C − 1 do
if ai−j = bi−j = ci−j = 0 then D ← Ø, C ← 1
end
foreach k € D do
if $k_i-2 \not= 0$ then ki−1 ← k_i, k_i-2 ← −k_i, k_i← 0
else k_i-2 ← k_i, k_i-3 ← −k_i, k_i← 0
end
while C > 0 do
S ← Ø (S)
S ← S + (aiP + biQ + ciR)
i ← i − 1 , C ← C − 1
end
end
return S

Three term simultaneous scalar multiplication for Koblitz curves.

previously described. The "Storage" column indicates the number of offline/online points needed in the precomputation stage, exclusive of the accumulator point for the result. The unmodified OpenSSL "Extract+Verify" performance of 25.461ms is rather disappointing, but indicative of the significant cost of using affine coordinates instead of projective coordinates.

TABLE II

SOFTWARE IMPLEMENTATION RESULTS, OPENSSL SOFTWARE TIMINGS.

With the unmodified OpenSSL, it is indeed faster simply to perform the extraction first then the verification (12.503ms), which is surprising. With the modified OpenSSL, the performance of signature generation is clearly improved, but could be improved even further with the use of a wider width τ NAF and offline precomputation (for example τ NAF_4). However, as expected the bottleneck is still clearly the verifica-tion. All of the modifications presented here significantly speed up these operations for OpenSSL. These timings should also scale accordingly with an increase in CPU speed, number of cores, and CPU size (e.g. 32 to 64-bit CPU). We also note that, although OpenSSL is used as a basis, significant linear speed increase is possible with the use of custom field arithmetic.

IV. CONCLUSION

In this paper, we have described an efficient method of implementing cryptography for PLA in datagram. A software implementation was presented, using the well-known OpenSSL distribution as a base; this was chosen in an attempt to make PLA more accessible. Extentions were provided to the standard OpenSSL API to support Koblitz curve operations and implicit certificates, as well as implementations of these API extensions which achieve significant speedup in signature generation as well as signature verification. The provided simultaneous public key extraction and signature verification has particularly competitive performance. As mentioned, even greater performance would be possible by providing custom finite field arithmetic, or with the use of custom hardware; for example, the corresponding hardware implementation reaches an impressive 166K combined extractions and verifications per second. These results have been achieved with PLA in mind— however, the provided implementations have a wide-range of other practical applications, wherever high-speed and compact elliptic curve cryptography is needed.

REFERENCES

[1] N. Koblitz, "Elliptic curve cryptosystems," Math. Comp., vol. 48, no. 177, pp. 203–209, 1987.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in Advances in cryptology—CRYPTO '85, ser. Lecture Notes in Comput. Sci., 1986, vol. 218, pp. 417–426.

[3] W. Meier and O. Staffelbach, "Efficient multiplication on certain non-supersingular elliptic curves," in Advances in cryptology—CRYPTO '92, ser. Lecture Notes in Comput. Sci. Springer-Verlag, 1993, vol. 740, pp.333–344.

[4] J. A. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," in Advances in cryptology—CRYPTO '97, ser. Lecture Notes in Comput. Sci. Springer-Verlag, 1997, vol. 1294, pp. 357–371.

[5] J. A. Solinas, "Efficient arithmetic on Koblitz curves," Des. Codes Cryptogr., vol. 19, no. 2-3, pp. 195–249, 2000.

[6] NIST, "Digital signature standard (DSS)," National Institute of Standards and Technology, FIPS PUB 186-2 (+ Change Notice), Jan. 2000.

[7] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," J. Cryptology, vol. 14, no. 4, pp. 255–293, 2001.

[8] D. Hankerson, A. J. Menezes, and S. A. Vanstone,

Guide to Elliptic Curve Cryptography. New York: Springer-Verlag, 2004.

[9] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software implementa-tion of elliptic curve cryptography over binary fields," in Cryptographic hardware and embedded systems—CHES '00. Springer, 2000, vol. 1965, pp. 243–267.

[10] A. Shamir, "How to check modular exponentiation," Presented at the rump session of EUROCRYPT '97. Konstanz, Germany, May 1997.

[11] J. A. Solinas, "Low-weight binary representations for pairs of integers," Centre for Applied Cryptographic Research, University of Waterloo, Canada, Tech. Rep. CORR 2001-41, 2001.

[12] M. Girault, "Self-certified public keys," in Advances in cryptology— EUROCRYPT '91, ser. Lecture Notes in Comput. Sci. Berlin: Springer, 1992, vol. 547, pp. 490–497.

[13] K. Nyberg and R. A. Rueppel, "A new signature scheme based on the DSA giving message recovery," in CCS '93: Proceedings of the 1st ACM conference on Computer and communications security. New York, NY, USA: ACM, 1993, pp. 58–61.

[14] G. Ateniese and B. de Medeiros, "A provably secure Nyberg-Rueppel signature variant with applications," Cryptology ePrint Archive, Report 2004/093, 2004, http://eprint.iacr.org/.

[15] B. B. Brumley and K. Nyberg, "Differential properties of elliptic curves and blind signatures," in Information Security, 10th International Conference—ISC '07, ser. Lecture Notes in Computer Science, vol. 4779. Springer-Verlag, 2007, pp. 376–389.

[16] K. Nyberg, "Differentially uniform mappings for cryptography," in Ad-vances in cryptology— EUROCRYPT '93, ser. Lecture Notes in Comput. Sci. Berlin: Springer, 1994, vol. 765, pp. 55–64.

[17] B. B. Brumley, "Efficient three-term simultaneous elliptic scalar multi-plication with applications," in Proceedings of the 11th Nordic Workshop on Secure IT Systems—NordSec '06, V. Fak,˚ Ed., Linkoping,¨ Sweden, Oct. 2006, pp. 105–116.

[18] M. Cox, R. Engelschall, S. Henson, and B. Laurie, *"The OpenSSL Project," http://www.openssl.org/.*

[19] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," Math. Comp., vol. 48, no. 177, pp. 243–264, 1987.

[20] J. Lopez´ and R. Dahab, "Fast multiplication on elliptic curves over GF(2^m) without precomputation," in Cryptographic hardware and embedded systems—CHES '99, ser. Lecture Notes in Comput. Sci. Springer-Verlag, 1999, vol. 1717, pp. 316–327.