Implementing Cryptography for Packet Level Authentication

Billy Bob Brumley
Department of Information and Computer Science
Helsinki University of Technology
P.O.Box 5400, FI-02015 TKK, Finland
billy.brumley@hut.fi

***Abstract*—Packet Level Authentication (PLA) is a novel countermeasure against distributed denial-of-service attacks. Each packet sent across a network has a digital signature and public key attached to it, allowing each hop along the route to verify the authenticity of packets. This requires high-speed elliptic curve cryptography (ECC) to improve throughput. In this paper, we present a software solution of cryptography for PLA using the combination of Koblitz curves to increase throughput and implicit certificates to decrease storage and computation overhead. A software implementation is presented, built on OpenSSL libraries and extending the OpenSSL 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.**

***Index Terms*—Elliptic curve cryptography, Koblitz curves, digital signatures, implicit certificates.**

I. Introduction

As Hardin noted [1] in "The Tragedy of the Commons", 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. Attacks such as (distributed) denial-of-service, packet spoofing, etc. are widespread. New and more efficient countermeasures are constantly required to protect against such attacks. *Packet Level Authentication* (PLA) [2] is one such countermeasure that 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 certified 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 also does not require previous communication between the sender and the receiver. This is different from other end-to-end solutions, such as IPSec, where 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.

Proceedings of The 2008 International Conference on Security & Management—SAM'08, CSREA Press, Arabnia, Aissi, Bedworth, Eds., pp. 475–480.

To implement PLA, signatures must be able to be 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. We provide OpenSSL API extensions for high-speed Koblitz curve support and implicit certificates. Results from a software implementation of these OpenSSL API extensions are provided. The result is high-speed, compact, and secure elliptic curve cryptography, for use not only with PLA, but for the OpenSSL community and elliptic curve cryptography in general.

We begin in Sec. II with a description of cryptography used in PLA, providing background on Koblitz curves, digital signatures, and implicit certificates. The software implementation is described in Sec. III and results given. We conclude in Sec. IV.

II. Cryptography for PLA

The cryptography parameters for PLA have been 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 [3] and implicit certificates. This results in fast and compact cryptography; a detailed description follows.

A. Koblitz Curves

Koblitz curves [3] are elliptic curves over $\mathbb{F}_2$ defined by the equation

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

The set of all (x, y) solutions over $\mathbb{F}_{2^m}$ along with the identity element $\mathcal{O}$, or point-at-infinity, form an abelian group with point addition (doubling) that can be used for cryptographic operations [4], [5]. These curves are of particular interest in high-speed cryptography as they admit the efficient endomorphism $\phi : E_a(\mathbb{F}_{2^m}) \rightarrow E_a(\mathbb{F}_{2^m})$ defined by

$$\phi(x, y) \mapsto (x^2, y^2),\ \phi(\mathcal{O}) \mapsto \mathcal{O} \tag{2}$$

called the *Frobenius endomorphism*. Squaring is a linear operation in $\mathbb{F}_{2^m}$ so this endomorphism can be applied extremely efficiently. It can be shown from the point addition formula that ϕ satisfies the characteristic polynomial

$$P(T) = T^2 - \mu T + 2 \text{ where } \mu = (-1)^{1-a} \tag{3}$$

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 $\mathbb{Z}[\tau]$ and expressed as the sum (and difference) of powers of τ, a base-τ expansion:

$$k = \sum_{i=0} c_i \tau^i \tag{4}$$

where c_i 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:

$$kP = \sum_{i=0} c_i \phi^i(P) \tag{5}$$

The length of such a straightforward expansion is twice that of the binary expansion, leading to twice the number of point additions on average. To shorten the length of the base-τ expansion, scalars are reduced using the fact [6] that $\phi^m(P) = P$ thus $\tau^m - 1 \equiv 0 \pmod{r}$. After such reduction, the resulting base-τ expansion has the same average length and weight as the binary expansion, and scalar multiplication is much faster as all of the point doublings are traded for applications of ϕ, a very fast operation compared to point doubling. This is the reason that Koblitz curves are often used when high-speed elliptic curve cryptography is needed.

To reduce the weight and thus the number of point additions, τ-adic analogues of width-w *Non-Adjacent Form* or NAF_w denoted τNAF_w exist; see [7], [8] for details.

For PLA, currently the standardized Koblitz curve K-163 [9] is used. With K-163, $m = 163$, $a = 1$ and $r \approx 2^{163}$. For comparison, this provides an equivalent level of 80 bits of symmetric cryptography security or 1024 bits compared to DSA/RSA[10]. 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 [11], [12].

B. Digital Signatures

The Elliptic Curve Digital Signature Algorithm (ECDSA) [9] is a widely used signature scheme that uses elliptic curves. A few domain parameters are first agreed upon. Elliptic curve E is chosen with base point generator G of prime order r where $r \mid \#E$. A collision-resistant hash

ECDSA SIGN
1) $f \leftarrow \text{H}(m)$
2) $k \leftarrow_R \mathbb{Z}_r^*$
3) $(x, y) \leftarrow kG$
4) $c \leftarrow x \mod r$
5) $d \leftarrow k^{-1}(f + cw) \mod r$
6) Return (c, d)

ECDSA VERIFY
1) $f \leftarrow \text{H}(m)$
2) $t \leftarrow d^{-1} \mod r$
3) $u \leftarrow ft \mod r$, $v \leftarrow ct \mod r$
4) $(x, y) \leftarrow uG + vW$
5) Return "Valid" iff $x \equiv c \pmod{r}$ else "Invalid"

Fig. 1. ECDSA signature and verification.

function H is also agreed upon. The procedure for signature generation and verification are given in in Fig. 1. A user generates a private key $w \in_R \mathbb{Z}_r^*$ and public key $W = wG$. To sign a message m, a user executes ECDSA SIGN. To verify the signature (c, d) on a message m, given public key W a user executes ECDSA VERIFY.

Signatures are of size $2r$ and public keys consist of points, so a pair (x, y). These keys can be compressed to just the x-coordinate and a compression bit; to decompress the point, given an x-coordinate the elliptic curve equation is solved for y, yielding at most two solutions—the compression bit tells which solution to use. Hence for Koblitz curves, public keys are stored using $m + 1$ bits. We will refer to the functions for point compression and decompression as COMPRESS and DECOMPRESS, respectively.

Signature generation requires one scalar multiplication by a fixed point (G), while signature verification requires two scalar multiplications: one by a fixed point (G) and one by an arbitrary point (W, the public key). More emphasis is thus put on speeding up verification rather than signing. As only the sum of the two points is needed in verification, it is common to carry out the scalar multiplications simultaneously using Shamir's trick [13]. Using some simple precomputation, this cuts the number of point doublings in half, but the joint expansion has slightly higher weight than a single expansion hence the number of point additions increases slightly when comparing a single expansion to a joint expansion. For example, a single scalar in NAF_2 (or τNAF_2) form has average weight 1/3; the joint weight of two scalars in Joint Sparse Form (JSF) [14] (or τJSF [15]) has average weight 1/2. In either case, the number of point additions and doublings (or applications of ϕ for Koblitz curves) when performing simultaneous scalar multiplication is significantly less than when processing the the scalar multiplications individually.

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 [16] 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 impersonation can 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 [17] is given in [18]. We presented the modified version from [19] which omits the proof-of-knowledge step. It reaches Trust Level 3 by blinding TTP from the user's private key.

Setup. Elliptic curve E is chosen with base point generator G of prime order r where $r \mid \#E$. TTP generates a domain private key $s_T \in_R \mathbb{Z}_r^*$ and domain public key $W_T = s_T G$. TTP then publishes W_T.

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

$$\begin{aligned} \text{TTP} \longleftarrow \text{Alice: } & k_A G \\ \text{TTP: } & (\overline{r}_A, b_A) = \text{COMPRESS}(k_A G + k_T G) \\ & r_A = \overline{r}_A + \text{H}(ID_A) \\ & \overline{s}_A = k_T - r_A s_T \pmod r \\ \text{Alice} \longleftarrow \text{TTP: } & (r_A, b_A, \overline{s}_A) \end{aligned} \tag{6}$$

Alice's private key is $s_A = k_A + \overline{s}_A \pmod r$.

Extract. To extract Alice's public key $W_A = s_A G$ on identity ID_A given public values (r_A, b_A), Bob calculates

$$W_A = \text{DECOMPRESS}(r_A - \text{H}(ID_A), b_A) - r_A W_T \tag{7}$$

The extracted public key is correct ($W_A = s_A G$):

$$\begin{aligned} W_A &= \text{DECOMPRESS}(r_A - \text{H}(ID_A), b_A) - r_A W_T \\ &= \text{DECOMPRESS}(\overline{r}_A + \text{H}(ID_A) - \text{H}(ID_A), b_A) - r_A W_T \\ &= k_A G + k_T G - r_A s_T G = (k_A + k_T - r_A s_T)G \\ &= (k_A + \overline{s}_A)G = s_A G \end{aligned}$$

2) Attempting Impersonation Attacks: In contrast to the key issuing protocol in [18], no proof-of-knowledge is performed by Alice in (6); TTP has no guarantee that Alice knows the discrete log of $k_A G$. 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 $d \in \mathbb{Z}_r$ such that the following equation holds:

$$[(k_A + d)G + k_T G]_x + \text{H}(ID_M) = [k_A G + k_T G]_x + \text{H}(ID_A).$$

Letting k denote $k_A + k_T$ (k is as random as k_T is), this becomes

$$[(k + d)G]_x - [kG]_x = \text{H}(ID_A) - \text{H}(ID_M). \tag{8}$$

In general, the ability to find such a difference is linked to the *differential uniformity* [20] 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. Fast Verifications using Implicit Certificates

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

$$uG + v(\text{DECOMPRESS}(r_A - \text{H}(ID_A), b_A) - r_A W_T)$$

and denoting the resulting point from decompression as D,

$$uG + v(D - r_A W_T) = uG + vD - v r_A W_T. \tag{9}$$

Therefore we first perform the point decompression, then calculate the value $-vr_A \mod r$ and perform the three needed scalar multiplications simultaneously similarly as with Shamir's trick. The online precomputation requirement increases 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 multiplications 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 [22] 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 arguments, 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 (using *interleaving* [11] 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 $1\mathsf{S} + 2\mathsf{M} + 1\mathsf{I}$ where $\mathsf{S}, \mathsf{M}, \mathsf{I}$ 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 [23] using projective coordinates from [24] that eliminates the need for precomputation.

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

To summarize, for PLA and Koblitz curve K-163 [9] 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. Extending OpenSSL

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. López-Dahab (LD) [25] projective coordinates were implemented, resulting in a point addition cost of $8\mathsf{M}+5\mathsf{S}$ using `ec_GF2m_ld_add` and point doubling $4\mathsf{M} + 5\mathsf{S}$ using `ec_GF2m_ld_dbl`.

OpenSSL does not have any special methods for Koblitz curves—they are treated as normal binary curves. We extend

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

OpenSSL to support Koblitz curve-specific functions. We implemented a function `compute_koblitz_reduce` that performs reduction of scalars as described in Sec. II-A—it is important call this function to shorten the length of scalars before scalar multiplication, otherwise the operation will take roughly twice as long. An analogous function `compute_wNAF_tau` is implemented that generates width$-w$ τNAFs as described in [8]. The function `ec_GF2m_koblitz_tau` applies the Frobenius (2)—in the case of projective coordinates, a minor cost of 3S.

For scalar multiplication using Koblitz curves, we implement 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* [11] method on the τNAF_2 is used, with a window width of $w = 3$; for two or three points, a method using Shamir's trick [13] with τ-adic scalar recoding [26] 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 τJSF [15]. 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 [23] for simultaneous inversion. This would allow the point additions 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.

C. Implementation Results

The implementation was done on an AMD Athlon Thunderbird 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 modifications to support implicit certificates and simultaneous key extraction and signature verification ("Extract+Verify"). The "Modified OpenSSL" section includes the entire implementation 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.

Input: ℓ-bit expansions a, b, c in τNAF_2, points $P, Q, R \in E_1(\mathbb{F}_{2^m})$
Output: $aP + bQ + cR$
Precompute $xP + yQ + zR\ \forall\ x, y, z \in \{-1, 0, 1\}$
$S \leftarrow \infty$, $i \leftarrow \ell - 1$
while $i \geq 0$ **do**
  $D \leftarrow \{a, b, c\}, C \leftarrow 1$
  foreach $k \in D$ **do**
    if $k_i = 0$ **then** $D \leftarrow D \setminus \{k\}$
    else if $k_i + k_{i-2} = \pm 2$ **then** $C \leftarrow \max(2, C)$
    else if $k_i + k_{i-3} = \pm 2$ **then** $C \leftarrow \max(3, C)$
    else $D \leftarrow \emptyset, C \leftarrow 1$
  end
  for $j \leftarrow 1$ **to** $C - 1$ **do**
    if $a_{i-j} = b_{i-j} = c_{i-j} = 0$ **then** $D \leftarrow \emptyset, C \leftarrow 1$
  end
  foreach $k \in D$ **do**
    if $k_{i-2} \neq 0$ **then** $k_{i-1} \leftarrow k_i, k_{i-2} \leftarrow -k_i, k_i \leftarrow 0$
    else $k_{i-2} \leftarrow -k_i, k_{i-3} \leftarrow -k_i, k_i \leftarrow 0$
  end
  while $C > 0$ **do**
    $S \leftarrow \phi(S)$
    $S \leftarrow S + (a_i P + b_i Q + c_i R)$
    $i \leftarrow i - 1$, $C \leftarrow C - 1$
  end
end
return S

Fig. 2. Three term simultaneous scalar multiplication for Koblitz curves.

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

TABLE II
SOFTWARE IMPLEMENTATION RESULTS, OPENSSL SOFTWARE TIMINGS.

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

coordinates. 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 τNAF and offline precomputation (for example $\tau\mathrm{NAF}_4$). However, as expected the bottleneck is still clearly the verification. 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. 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 [27] 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.

Readers interested in deploying PLA are encouraged to visit the project website at http://www.tcs.hut.fi/Software/PLA/new/.

References

[1] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968.

[2] C. Candolin, J. Lundberg, and H. Kari, "Packet level authentication in military networks," in *Proceedings of the 6th Australian Information Warfare & IT Security Conference*, Geelong, Australia, Nov. 2005.

[3] N. Koblitz, "CM-curves with good cryptographic properties," in *Advances in cryptology—CRYPTO '91*, ser. Lecture Notes in Comput. Sci. Springer-Verlag, 1992, vol. 576, pp. 279–287.

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

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] D. Hankerson, A. J. Menezes, and S. A. Vanstone, *Guide to Elliptic Curve Cryptography*. New York: Springer-Verlag, 2004.

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

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

[14] 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.

[15] M. Ciet, T. Lange, F. Sica, and J.-J. Quisquater, "Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms," in *Advances in cryptology—EUROCRYPT '03*, ser. Lecture Notes in Comput. Sci. Berlin: Springer, 2003, vol. 2656, pp. 387–400.

[16] 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.

[17] 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.

[18] 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/.

[19] 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.

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

[21] B. B. Brumley, "Efficient three-term simultaneous elliptic scalar multiplication with applications," in *Proceedings of the 11th Nordic Workshop on Secure IT Systems—NordSec '06*, V. Fåk, Ed., Linköping, Sweden, Oct. 2006, pp. 105–116.

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

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

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

[25] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in $\mathrm{GF}(2^n)$," in *Selected areas in cryptography—SAC '98*, ser. Lecture Notes in Comput. Sci. Berlin: Springer, 1999, vol. 1556, pp. 201–212.

[26] B. B. Brumley, "Left-to-right signed-bit τ-adic representations of n integers (short paper)," in *Information and Communications Security, 8th International Conference—ICICS '06*, ser. Lecture Notes in Computer Science, vol. 4307. Springer-Verlag, 2006, pp. 469–478.

[27] K. U. Järvinen, J. Forsten, and J. Skyttä, "FPGA design of self-certified signature verification on Koblitz curves," in *Cryptographic hardware and embedded systems—CHES '07*, ser. Lecture Notes in Comput. Sci. Berlin: Springer, 2007, vol. 4727, pp. 256–271.