certicom™

Speed-ups of Elliptic Curve-Based Schemes

René Struik
Certicom Research
e-mail: rstruik@certicom.com

FIELDS

Workshop on
New Directions in
Cryptography
June 25-27, 2008

Part I – Accelerated Verification of ECDSA Signatures

René Struik
Certicom Research
e-mail: rstruik@certicom.com

Joint work with A. Antipa, D.R. Brown, R. Gallant, R. Lambert, S.A. Vanstone

Workshop on New Directions in Cryptography
June 25-27, 2008

Outline

- ECDSA signature scheme
- Fast ECDSA signature scheme
- Computational aspects
  - Simultaneous multiplication
  - Extended Euclidean Algorithm
- Examples
  - Fast ECDSA verification
  - ECDSA verification
  - Comparison with RSA signatures
- Generalizations
- Conclusions

ECDSA signature scheme

System-wide parameters

Elliptic curve of prime order n with generator G. Hash function h.

Initial set-up

Signer A selects private key $d \in [1,n-1]$ and publishes its public key $Q = dG$.

Signature generation

INPUT: Message m, private key d.
OUTPUT: Signature (r, s).

ACTIONS:

1. Compute $e := h(m)$.
2. Select random $k \in [1,n-1]$.
3. Compute $R := kG$ and map R to r.
4. Compute $s := k^{-1}(e + d\,r) \bmod n$.
5. If $r, s \in [1,n-1]$, return (r, s); otherwise, go to Step 2.

Signature verification

INPUT: Message m, signature (r, s); Public signing key Q of Alice.
OUTPUT: Accept or reject signature.

ACTIONS:

1. Compute $e := h(m)$.
2. Check that $r, s \in [1,n-1]$. If verification fails, return 'reject'.
3. Compute $R' := s^{-1}(e\,G + r\,Q)$.
4. Check that R' maps to r. If verification succeeds, return 'accept'; otherwise return 'reject'.

Non-repudiation: Verifier knows the true identity of the signing party, since the public signing key Q is bound to signing party Alice.

Fast ECDSA signature scheme

System-wide parameters

Elliptic curve of prime order n with generator G. Hash function h.

Initial set-up

Signer A selects private key $d \in [1,n-1]$ and publishes its public key $Q = dG$.

Signature generation

INPUT: Message m, private key d.
OUTPUT: Signature (R, s).

ACTIONS:

1. Compute e:= $h(m)$.
2. Select random $k \in [1,n-1]$.
3. Compute $R := kG$ and map R to r.
4. Compute s:= $k^{-1}(e + d\,r) \bmod n$.
5. If $r, s \in [1,n-1]$, return (R, s); otherwise, go to Step 2.

Signature verification

INPUT: Message m, signature (R, s); Public signing key Q of Alice.
OUTPUT: Accept or reject signature.

ACTIONS:

1. Compute e:= $h(m)$.
2. Map R to r.
3. Check that $r, s \in [1,n-1]$. If verification fails, return 'reject'.
4. Check that $R = s^{-1}(e\,G + r\,Q)$. If verification succeeds, return 'accept'; otherwise return 'reject'.

Non-repudiation: Verifier knows the true identity of the signing party, since the public signing key Q is bound to signing party Alice.

Fast ECDSA signature scheme

Computational aspects

Ordinary signature verification	Fast signature verification
ACTIONS: … **3. Compute** $R' := (e\, s^{-1})\, G + (r\, s^{-1})\, Q$. **4. Check that** R' **maps to** r. …	**ACTIONS:** … **2. Map** R **to** r. **4. Check that** $R = (e\, s^{-1})\, G + (r\, s^{-1})\, Q$. …

Ordinary signature verification

Compute expression $R' := (e\, s^{-1})\, G + (r\, s^{-1})\, Q$.

Cost: *full-size* linear combination of *known* point G and *unknown* point Q.

Fast signature verification

Evaluate expression $\Delta := s^{-1}\,(e\, G + r\, Q) - R$ and check that $\Delta = O$.

Cost: *full-size* linear combination of *known* point G and *unknown* point Q.

Seemingly no computational advantages over traditional approach … ☹

Computational aspects (1)

One can do better, though! ☺

Fast signature verification

Evaluate expression $\Delta := (e\,s^{-1})\,G + (r\,s^{-1})\,Q - R$ and check that $\Delta = O$.

Equivalent test

Check that $\mu\,\Delta := (\mu\,e\,s^{-1})\,G + (\mu\,r\,s^{-1})\,Q - \mu\,R = O$ for any $\mu \in [1, n-1]$.

or:

Check that $\mu\,\Delta := (\mu\,e\,s^{-1})\,G + \lambda\,Q - \mu\,R = O$, where $r/s \equiv \lambda/\mu \pmod{n}$.

Optimum choice

Write $r/s \equiv \lambda/\mu \pmod{n}$, where λ and μ have size *half* the bit-length of n.

Note: This can be done efficiently using the Extended Euclidean Algorithm.

Why speed-up?

Speed-up due to getting rid of half of so-called point doubles.

Computational aspects (2)

Fast signature verification

Check that $\mu\,\Delta := (\mu\, e\, s^{-1})\, G + \lambda\, Q - \mu\, R = O$, where $r / s \equiv \lambda / \mu \pmod{n}$ and where λ and μ have size *half* the bit-length of n.

Details:

Pre-compute $G_1 := t\, G$, where $t \approx \sqrt{n}$. Let $G_0 := G$.

Write $r / s \equiv \lambda / \mu \pmod{n}$, where λ and μ have size *half* the bit-length of n.

Write $\mu\, e\, s^{-1} \equiv \alpha_0 + \alpha_1\, t \pmod{n}$, where α_0, α_1 have size half the bit-length of n.

Evaluate
$$\begin{aligned}\mu\Delta &:= (\mu\, e\, s^{-1})\, G + \lambda\, Q - \mu\, R \\ &= \alpha_0\, G_0 + \alpha_1\, G_1 + \lambda\, Q - \mu\, R\end{aligned}$$

<u>Cost:</u> *half-size* combination of *known* points G_0, G_1 and *unknown* points Q, R.

Ordinary signature verification

Compute expression $R' := (e\, s^{-1})\, G + (r\, s^{-1})\, Q$.

<u>Cost:</u> *full-size* linear combination of *known* point G and *unknown* point Q.

Computational aspects (3)

Optimum choice

Write $r / s \equiv \lambda / \mu \pmod{n}$, where λ and μ have size *half* the bit-length of n.

This can be done efficiently using the Extended Euclidean Algorithm.

Extended Euclidean Algorithm (EEA)

INPUT: Positive integers a and n with $a \leq n$.

OUTPUT: $d = \gcd(a, n)$ and integers x, y satisfying $a\,x + n\,y = d$.

ACTIONS:

1. $(u, v) \leftarrow (a, n)$; $X \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$;
2. while $u \neq 0$ do
   {
   $q \leftarrow v$ div u; $(u, v) \leftarrow (v \bmod u, u)$; $X \leftarrow \begin{pmatrix} -q & 1 \\ 1 & 0 \end{pmatrix} X$
   }
3. $(d, x, y) \leftarrow (v, x_{21}, x_{22})$.

Invariant:

$$a\,x_{11} + n\,x_{12} = u$$
$$a\,x_{21} + n\,x_{22} = v$$

Let $a := r\,s^{-1} \pmod{n}$.

Use Ext. Euclidean Algorithm to compute $\gcd(a, n)$. (which is 1, since n is prime.)

Abort algorithm once $u < \sqrt{n}$. (Most likely, $|x_{11}|$ is also close to $\sqrt{n}$.)

Set $\lambda := u$ and $\mu := x_{11}$.

Example

Verification cost ECDSA scheme vs. Fast ECDSA scheme

- Curve: NIST prime curve P-384 with 192-bit security (Suite B)
- Integer representation: NAF, joint sparse form (JSF)
- Coordinate system: Jacobian coordinates

P-384 curve	ECDSA Verify	
ECC operations	Ordinary	Fast
– Add	194	196
– Double	384	192
– Total[1]	459	328

[1]Normalized (double/add ratio: 0.69)

RIM Blackberry[2]	221 ms	158 ms

[2]Platform: ARM7TDMI (50 MHz)

Speed-up cost Fast ECDSA verify

compared to ordinary approach: 1.4x

ECDSA vs. Fast ECDSA

Security of Fast ECDSA

Both schemes are equally secure: ECDSA has signature (r, s) if and only if Fast ECDSA has signature (R, s) where R maps to r.

ECDSA signature verification

- Convert ECDSA signature (r, s) to Fast ECDSA signature (R, s)
- Verify Fast ECDSA signature (R, s)

Note:

- Conversion generally yields *pair* (R, $-R$) of *candidate points* that map to r.
- Verification involves trying out all those candidate points not discarded based on some side constraints (the so-called *admissible points*).

How to ensure only one admissible point:

- Generate ECDSA signature with k such that y-coordinate of $R:=kG$ can be prescribed. (If necessary, change the sign of k.)
- Use the fact that (r, s) is a valid ECDSA signature if and only if (r, $-s$) is.

Cost of signature verification

Verification cost of ECDSA signature vs. RSA signatures

- RSA: public exponent $e = 2^{16}+1$
- ECDSA: NIST prime curves
- Platform: HP iPAQ 3950, Intel PXA250 processor (400 MHz)

Security level (bits)	Verification cost (ms)			Ratio fast ECDSA verify vs. RSA verify
	RSA[2]	ECDSA		
		ordinary[2]	fast[3]	
80	1.4	4.0	2.9	0.5x faster
112	5.2	7.7	5.5	0.9x faster
128	11.0	11.8	8.4	1.3x faster
192	65.8	32.9	23.5	2.8x faster
256	285.0	73.2	52.3	5.4x faster

[1]Excluding (fixed) overhead of identification data
[2]Certicom Security Builder [3]Estimate

Conclusion

Efficiency advantage of RSA signatures over ECDSA signatures is vanishing

Generalizations

Method for accelerated signature verification works in more general setting than presented here:

- Verification:
  - Fast ECDSA signature verification when more than one multiple of the signer's public key Q is available (e.g., included in 'fat' certificate)
  - Verification of any elliptic curve equation involving an unknown point
  - Verification of any elliptic curve equation involving more than one unknown point (use lattice base reduction in low-dimensional lattice)
- Algebraic group:
  - Operations in other algebraic structures (including hyper-elliptic curves, identity-based crypto systems)

Conclusions

Fast ECDSA signature scheme attractive:

- Security: Same security as original ECDSA signature scheme
- Efficiency: Considerable speed-up possible for non-Koblitz curves
  - NIST prime curves, ‘Suite B’ curves, Brainpool curves: 40% speed-up
  - NIST random binary curves: 40% speed-up

Efficiency results applicable to ordinary ECDSA signature scheme:

- ECDSA and Fast ECDSA have same cost if only 1 admissible point
  - Append 1 bit of side info to ECDSA signature to distinguish (R, $-R$)
  - Agree on particular way of generating ECDSA signatures such that only one of points R and $–R$ is admissible
- ECDSA can still use Fast ECDSA if more than 1 admissible point
  - Roughly 8% average speed-up for curves mentioned above

Efficiency advantage of RSA signatures over ECDSA signatures is vanishing

Part II – Combined Verification and Key Computation

René Struik
Certicom Research
e-mail: rstruik@certicom.com

Workshop on New Directions in Cryptography
June 25-27, 2008

FIELDS

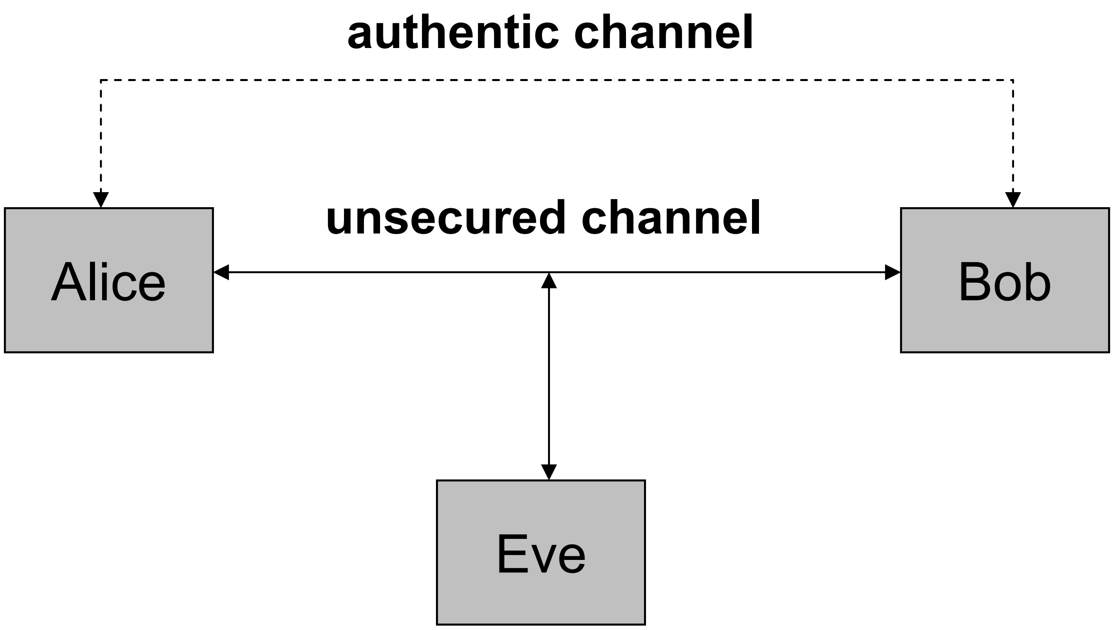
Outline

- Public key cryptography
  - Key agreement schemes
  - Signature schemes
- Computational aspects
  - Key computation
  - Certificate verification
  - Combined key computation and certificate verification
- Examples
  - Static Diffie-Hellman with ECDSA certificates
  - ECMQV with ECDSA certificates
  - Comparison with RSA certificates
- Generalizations
- Conclusions

Public key cryptography

Communication model

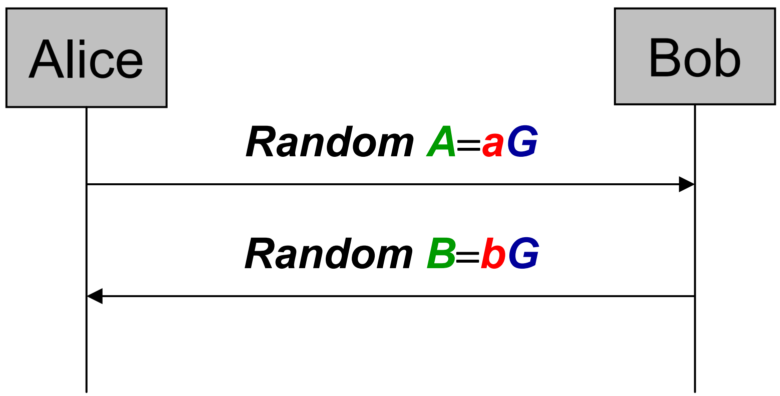
Communicating parties a priori share authentic information



Key agreement schemes

Anonymous Diffie-Hellman (ephemeral ECDH)

ACTIONS:
1. Select $a \in_R [1,n-1]$.
2. Compute $A := aG$.
3. Compute $K := aB$.



ACTIONS:
1. Select $b \in_R [1,n-1]$.
2. Compute $B := bG$.
3. Compute $K := bA$.

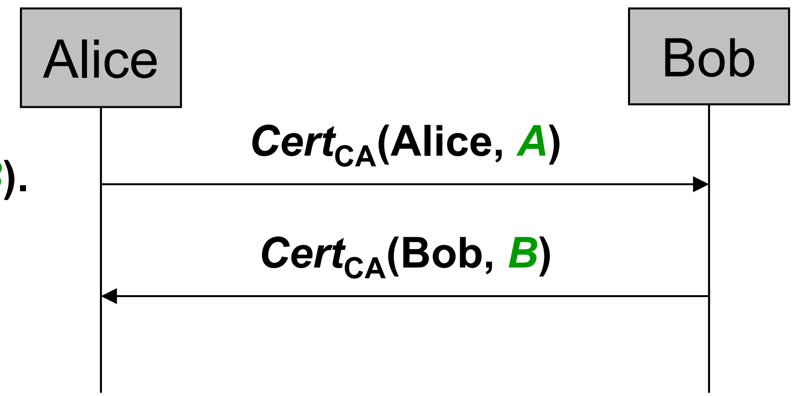
Properties

- Key agreement: Both parties arrive at same key K, since $K = abG = aB = bA$.
- No key authentication: Neither party knows the true identity of the key sharing party, since keys A and B are *not* bound to parties Alice and Bob.

Key agreement schemes

Authenticated Diffie-Hellman (static ECDH)

ACTIONS:
1. Verify Cert_{CA}**(Bob,** B**).**
2. Compute $K := aB$**.**



ACTIONS:
1. Verify Cert_{CA}**(Alice,** A**).**
2. Compute $K := bA$**.**

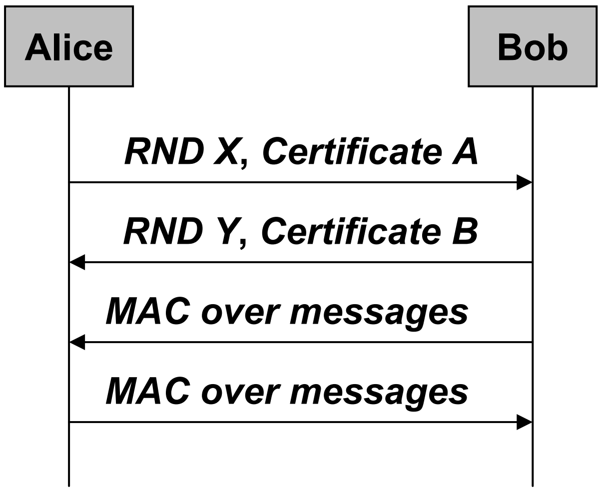
Properties

- Key agreement: Both parties arrive at same key K, since $K = abG = aB = bA$.
- Key authentication: Each party knows the true identity of the key sharing party, since keys A and B *are* bound to parties Alice and Bob.

Key agreement schemes

General protocol format

Step 1: Key contributions
Each party randomly generates a short-term (ephemeral) public key pair and communicates the ephemeral public key to the other party (but not the private key).
Step 2: Key establishment
Each party computes the shared key based on static and ephemeral public keys received from the other party and static and ephemeral private keys it generated itself.
Step 3: Key authentication
Each party verifies the authenticity of the static key of the other party.
Step 4: Key confirmation
Each party evidences possession of the shared key to the other party. This also confirms its true identity to the other party.



Key agreement schemes

Computational aspects

Offline fixed point multiplication

Step 1: Key contributions

Each party randomly generates a short-term (ephemeral) public key pair and communicates the ephemeral public key to the other party (but not the private key).

Step 2: Key establishment

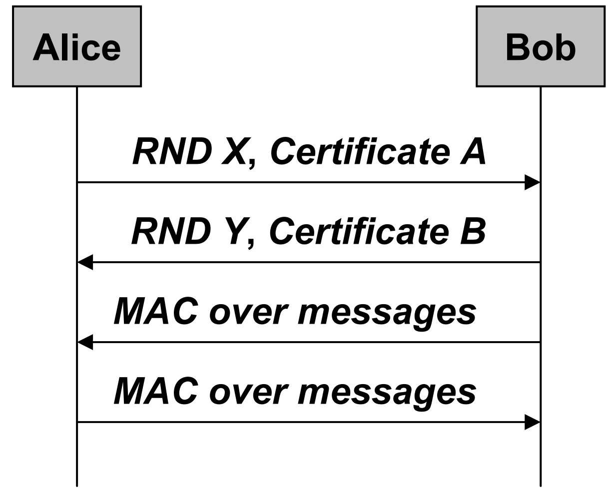
Each party computes the shared key based on static and ephemeral public keys received from the other party and static and ephemeral private keys it generated itself.

Step 3: Key authentication

Each party verifies the authenticity of the static key of the other party.

Step 4: Key confirmation

Each party evidences possession of the shared key to the other party. This also confirms its true identity to the other party.



Online variable point multiplication

Online verification of public key certificate

ECDSA signature scheme

ECDSA signature verification

INPUT: Message m, signature (r, s);
Public signing key Q of Alice.

OUTPUT: Accept or reject signature.

System-wide parameters

Elliptic curve with generator G.

Hash function h.

Ordinary signature verification

ACTIONS:

...

1. Compute $e := h(m)$.
2. Compute $R' := (e\, s^{-1})\, G + (r\, s^{-1})\, Q$.
3. Check that R' maps to r.

...

Fast signature verification

ACTIONS:

...

1. Compute $e := h(m)$.
2. Reconstruct R from r.
3. Check that $R = (e\, s^{-1})\, G + (r\, s^{-1})\, Q$.

...

ECDSA verification: Check equation $s^{-1}(e\, G + r\, Q) - R = O$.

Non-repudiation: Verifier knows the true identity of the signing party, since the public signing key Q is bound to signing party Alice.

Computational aspects (1)

Step 2: ECDH key computation (key establishment)

ACTIONS (Alice):
1. Verify $Cert_{CA}$(Bob,B).
2. Compute K:=aB.

Compute expression $K := aB$,

where a is Alice's private key;
B is Bob's public key (derived from his certificate).

Step 3: ECDSA certificate verification (key authentication)

Evaluate expression $s^{-1}(e\,G + r\,Q) - R = O$,

where e is hash value of certificate info (including Bob, B);
Q is public key of certificate authority;
(r, s) is ECDSA signature over certificate info.

Question: Can one combine these steps?

Answer: YES!

Computational aspects (2)

Step 2: ECDH key computation (key establishment)

Compute expression $K := aB$.

ACTIONS (Alice):
1. Verify $Cert_{CA}$(Bob,B).
2. Compute K:=aB.

Step 3: ECDSA certificate verification (key authentication)

Evaluate expression $\Delta := s^{-1}(e\,G + r\,Q) - R$ and check that $\Delta = O$.

Step 2 and Step 3 combined: Combined verification and key computation

Compute expression $K' := aB + \lambda\,(s^{-1}(e\,G + r\,Q) - R)$ instead.

Verification expression: $s^{-1}(e\,G + r\,Q) - R$
Randomizer: λ
Key expression: aB

More generally, compute $K' := K + \lambda\,\Delta$ instead.

Computational aspects (3)

Step 2 and Step 3 combined: Combined verification and key computation

Compute expression $K' := aB + \lambda\,(s^{-1}(e\,G + r\,Q) - R)$ instead.

Verification expression — $s^{-1}(e\,G + r\,Q) - R$

Randomizer — λ

Key expression — aB

More generally, compute $K' := K + \lambda\,\Delta$ instead.

Why does this work?

Alice can only compute K' correctly if certificate is 'correct' (i.e., $\Delta = O$); otherwise, K' is random (since then $\Delta \neq O$).

Property

Implicit key authentication: Each party knows the true identity of the key sharing party, if any, since keys A and B are bound to parties Alice and Bob and either party can only compute a shared key if that binding is 'correct'.

Computational aspects (4)

Step 2: ECDH key computation (key establishment)

Compute expression $K := aB$.

Cost: full-size multiple of *unknown* point B.

Step 3: ECDSA certificate verification (key authentication)

Check expression $s^{-1}(e\,G + r\,Q) = R$.

Cost: linear combination of *known* point G and *unknown* point Q.

Step 2 and Step 3 combined: Combined verification and key computation

Compute expression $K' := aB - \lambda\, R + (\lambda\, e\, s^{-1})\, G + (\lambda\, r\, s^{-1})\, Q$.

Cost: linear combination of *known* point G and *unknown* points B,Q, and R.

Why speed-up?

Speed-up due to getting rid of half of so-called point doubles.

Example (1)

Static ECDH with ECDSA certificates

- Curve: NIST prime curve P-384 with 192-bit security (Suite B)
- Integer representation: NAF, joint sparse form (JSF)
- Coordinate system: Jacobian coordinates

P-384 curve	ECDH key	ECDSA (incremental cost)		
		Separately		Combined with ECDH
ECC operations		Ordinary	Fast	
– Add	128	194	196	195
– Double	384	384	192	–
– Total[1]	393	459	328	195

[1]Normalized (double/add ratio: 0.69)

Speed-up incremental cost ECDSA verify

compared to separate approach: 2.4x (ordinary ECDSA verify)
1.7x (Fast ECDSA verify)

Example (2)

ECMQV with ECDSA certificates

- Curve: NIST prime curve P-384 with 192-bit security (Suite B)
- Integer representation: NAF, joint sparse form (JSF)
- Coordinate system: Jacobian coordinates

P-384 curve	ECMQV key	ECDSA (incremental cost)		
		Separately		Combined with ECMQV
ECC operations		Ordinary	Fast	
– Add	194	194	196	196
– Double	384	384	192	–
– Total[1]	459	459	328	196

[1]Normalized (double/add ratio: 0.69)

Speed-up incremental cost ECDSA verify

compared to separate approach: 2.3x (ordinary ECDSA verify)
1.7x (Fast ECDSA verify)

Example (3)

Static ECDH and ECMQV with ECDSA certificates

P-384 curve Total ECC operations[1]	Key computation	Key computation + ECDSA (total cost)		
		ECDSA separately		ECDSA combined
		Ordinary	Fast	
ECDH	393	852	721	588
ECMQV	459	918	787	655

[1]**Normalized (double/add ratio: 0.69)**

Speed-up total cost ECDH + ECDSA
compared to separate approach: +45% (ordinary ECDSA verify)
+23% (Fast ECDSA verify)

Speed-up total cost ECMQV + ECDSA
compared to separate approach: +40% (ordinary ECDSA verify)
+20% (Fast ECDSA verify)

Cost of certificate verification

Incremental verification cost of ECDSA certificates vs. RSA certificates

- RSA: public exponent $e = 2^{16}+1$
- ECDSA, ECDH: NIST prime curves
- Platform: HP iPAQ 3950, Intel PXA250 processor (400 MHz)

Security level (bits)	Certificate size[1] (bytes)		Ratio ECC/RSA certificates	Verify – incremental cost (ms)			Ratio ECDSA verify vs. RSA verify
	ECDSA	RSA		RSA[2]	ECDSA ordinary[2]	ECDSA combined[3]	
80	72	256	4x smaller	1.4	4.0	1.7	0.8x faster
112	84	512	6x smaller	5.2	7.7	3.2	1.6x faster
128	96	768	8x smaller	11.0	11.8	4.9	2.2x faster
192	144	1920	13x smaller	65.8	32.9	13.7	4.8x faster
256	198	3840	19x smaller	285.0	73.2	30.5	9.3x faster

[1]Excluding (fixed) overhead of identification data [2]Certicom Security Builder [3]Estimate

Conclusion

Efficiency advantage of RSA certificates with DH-based schemes is no more

Generalizations

Method for combining verification with key computation works in more general setting than presented here:

- Verification:
  - Verification of multiple ECDSA signatures (certificate chains)
  - Verification of any elliptic curve equation
  - Batch verification of multiple elliptic curve equations
- Key computation:
  - Key computation with ECDH-schemes in ANSI X9.63, NIST SP800-56a (including ECIES, Unified Model, STS, ECMQV, ElGamal encryption)
  - Computation of non-secret ECC point (if correctness can be checked)
  - Computation of multiple ECC points (if correctness can be checked)
- Algebraic group:
  - Operations in other algebraic structures (including hyper-elliptic curves, identity-based crypto systems)
- Side channel resistance:
  - Simple side channel resistance virtually for free

Conclusions

Combined computation of ECDH-key and ECDSA verification attractive:

- Security: Same security as underlying DH-based key agreement scheme or ECDSA signature scheme
- Efficiency: Considerable speed-up for all NIST prime curves
  - ECDH + ECDSA: up to 45% speed-up total online cost
  - ECMQV + ECDSA: up to 40% speed-up total online cost
  - ECDSA: up to 2.4x speed-up incremental ECDSA cost
- Implementation security: Simple side channel resistance virtually for free

Incremental cost of signature verification is the right metric:

- Efficiency advantage of RSA certificates with ECDH scheme is no more
  - Break-even point already at roughly 80-bit security level

Many generalizations possible…

Further reading

1. ANSI X9.63-2001, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American National Standard for Financial Services, American Bankers Association, November 20, 2001.
2. A. Antipa, D.R. Brown, R. Gallant, R. Lambert, R. Struik, S.A. Vanstone, 'Accelerated Verification of ECDSA Signatures,' in *Proceedings of Selected Areas in Cryptography – SAC2005*, B. Preneel, S. Tavares, Eds., Lecture Notes in Computer Science, Vol.~3897, pp. 307-318, New York: Springer, 2006.
3. M. Bellare, J.A. Garay, T. Rabin, 'Fast Batch Verification for Modular Exponentiation and Digital Signatures,' in *Proceedings of Advances in Cryptology – EUROCRYPT'98*, K. Nyberg, Ed., Lecture Notes in Computer Science, Vol. 1403, pp. 236-250, New York: Springer-Verlag, 1998.
4. FIPS Pub 186-2, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-2, US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, Maryland, USA, January 27, 2000.
5. D.R. Hankerson, A.J. Menezes, S.A. Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer, 2003.
6. NIST SP800-56a, *Recommendation for Pair-wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, March 8, 2007.
7. SEC4 – *Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV)*, Standards for Efficient Cryptography Group, Draft v0.9, November 14, 2007.
8. R. Struik, 'Combined Verifications and Key Computations,' draft.