Speed-ups of Elliptic Curve-Based Schemes

René Struijk
independent
e-mail: rstruijk.ext@gmail.com

Results based on work conducted at Certicom Research
Outline

• ECDSA signature scheme
• Fast ECDSA signature scheme
• Speed-ups:
  – ECDSA fast verification
  – ECDSA certificate verification and ECC-based key agreement (ECDH, ECMQV)
  – Batch ECDSA verification
• How to get from ECDSA to Fast ECDSA
• How an IETF standard could help
• IPR aspects

René Struik, e-mail: rstruik.ext@gmail.com
ECDSA signature scheme

<table>
<thead>
<tr>
<th>System-wide parameters</th>
<th>Initial set-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curve of prime order ( n ) with generator ( G ). Hash function ( h ).</td>
<td>Signer A selects private key ( d \in [1, n-1] ) and publishes its public key ( Q = dG ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signature generation</th>
<th>Signature verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT: Message ( m ), private key ( d ).</td>
<td>INPUT: Message ( m ), signature ((r, s)); Public signing key ( Q ) of Alice.</td>
</tr>
<tr>
<td>OUTPUT: Signature ((r, s)).</td>
<td>OUTPUT: Accept or reject signature.</td>
</tr>
</tbody>
</table>

**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 \( s := k^{-1}(e + d \cdot r) \) mod \( n \).
5. If \( r \notin [1, n-1] \) or \( s \notin [1, n-1] \), go to #2.
6. Return \((r, s)\).

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

René Struik, e-mail: rstruik.ext@gmail.com
# Fast ECDSA signature scheme

<table>
<thead>
<tr>
<th>System-wide parameters</th>
<th>Initial set-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curve of prime order ( n ) with generator ( G ). Hash function ( h ).</td>
<td>Signer A selects private key ( d \in [1,n-1] ) and publishes its public key ( Q = dG ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signature generation</th>
<th>Signature verification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT:</strong> Message ( m ), private key ( d ).</td>
<td><strong>INPUT:</strong> Message ( m ), signature ((R, s)); Public signing key ( Q ) of Alice.</td>
</tr>
<tr>
<td><strong>OUTPUT:</strong> Signature ((R, s)).</td>
<td><strong>OUTPUT:</strong> Accept or reject signature.</td>
</tr>
</tbody>
</table>

**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 \( s := k^{-1}(e + d \cdot r) \mod n \).
5. If \( r \notin [1,n-1] \) or \( s \notin [1,n-1] \), go to #2.
6. Return \((R, s)\).

**ACTIONS:**
1. If \( r \notin [1,n-1] \), return ‘reject’.
2. If \( s \notin [1,n-1] \), return ‘reject’.
3. Map \( R \) to \( r \).
4. Compute \( e := h(m) \).
5. Check that \( R = s^{-1}(e \cdot G + r \cdot Q) \).
   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.

*René Struik, e-mail: rstruik.ext@gmail.com*
Fast ECDSA and speed-ups

Speed-ups for prime curves and binary non-Koblitz curves:
– NIST prime curves, ‘Suite B’ curves, Brainpool curves, GOST (RFC 5832)
– NIST random binary curves

**Fast verification of ECDSA signatures ([2])**:  
40% speed-up compared to ordinary approach

**ECDSA certificate verification + Static ECDH/ECMQV ([7])**:  
Speed-up incremental cost ECDSA verify compared to separate approach:  
2.4x speed-up (compared to ordinary ECDSA verify)  
1.7x (compared to Fast ECDSA verify)  
Simple side channel resistance virtually for free

**Batch verification of ECDSA signatures ([3])**:  
Dependent on number of signatures involved

René Struik, e-mail: rstruik.ext@gmail.com
Part I –
Accelerated Verification of ECDSA Signatures

René Struik
independent
e-mail: rstruik.ext@gmail.com

Joint work with A. Antipa, D.R. Brown, R. Gallant, R. Lambert, S.A. Vanstone
Fast ECDSA signature scheme

Computational aspects

**Ordinary signature verification**

**ACTIONS:**

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

4. Check that $R'$ maps to $r$.

**Fast signature verification**

**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 $\Delta = O$.

by verifying instead

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

Cost: half-size combination of known points $G$, $G'$ and unknown points $Q$, $R$.

René Struik, e-mail: rstruik.ext@gmail.com
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

<table>
<thead>
<tr>
<th>P-384 curve</th>
<th>ECDSA Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC operations</td>
<td>Ordinary</td>
</tr>
<tr>
<td>– Add</td>
<td>194</td>
</tr>
<tr>
<td>– Double</td>
<td>384</td>
</tr>
<tr>
<td>– Total$^1$</td>
<td>459</td>
</tr>
</tbody>
</table>

$^1$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

René Struik, e-mail: rstruik.ext@gmail.com
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)

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>Verification cost (ms)</th>
<th>Ratio fast ECDSA verify vs. RSA verify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RSA² ordinary² fast³</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1.4 4.0 2.9</td>
<td>0.5x faster</td>
</tr>
<tr>
<td>112</td>
<td>5.2 7.7 5.5</td>
<td>0.9x faster</td>
</tr>
<tr>
<td>128</td>
<td>11.0 11.8 8.4</td>
<td>1.3x faster</td>
</tr>
<tr>
<td>192</td>
<td>65.8 32.9 23.5</td>
<td>2.8x faster</td>
</tr>
<tr>
<td>256</td>
<td>285.0 73.2 52.3</td>
<td>5.4x faster</td>
</tr>
</tbody>
</table>

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

Conclusion
Efficiency advantage of RSA signatures over ECDSA signatures is vanishing

René Struik, e-mail: rstruik.ext@gmail.com
Part II –
Combined Verification
and Key Computation

René Struik
independent
e-mail: rstruik.ext@gmail.com
Key agreement schemes

Authenticated Diffie-Hellman (static ECDH)

**ACTIONS:**
1. Verify $\text{Cert}_{CA}(Bob, B)$.

**ACTIONS:**
1. Verify $\text{Cert}_{CA}(Alice, A)$.
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.

René Struik, e-mail: rstruik.ext@gmail.com
Computational aspects (1)

Step 2: ECDH key computation (key establishment)

Compute expression $K := aB,$

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

Step 1: ECDSA certificate verification (key authentication)

Evaluate expression $s^{-1} (eG + rQ) - R = O,$

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!

René Struik, e-mail: rstruik.ext@gmail.com
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

<table>
<thead>
<tr>
<th>P-384 curve</th>
<th>ECDH key</th>
<th>ECDSA (incremental cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC operations</td>
<td>Separately</td>
<td>Combined with ECDH</td>
</tr>
<tr>
<td>Add</td>
<td>128</td>
<td>194</td>
</tr>
<tr>
<td>Double</td>
<td>384</td>
<td>384</td>
</tr>
<tr>
<td>Total¹</td>
<td>393</td>
<td>459</td>
</tr>
</tbody>
</table>

¹Normalized (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)

René Struik, e-mail: rstruiext@gmail.com
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)

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>Certificate size¹ (bytes)</th>
<th>Ratio ECC/RSA certificates</th>
<th>Verify – incremental cost (ms)</th>
<th>Ratio ECDSA verify vs. RSA verify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ECDSA</td>
<td>RSA</td>
<td></td>
<td>RSA²</td>
</tr>
<tr>
<td>80</td>
<td>72</td>
<td>256</td>
<td>4x smaller</td>
<td>1.4</td>
</tr>
<tr>
<td>112</td>
<td>84</td>
<td>512</td>
<td>6x smaller</td>
<td>5.2</td>
</tr>
<tr>
<td>128</td>
<td>96</td>
<td>768</td>
<td>8x smaller</td>
<td>11.0</td>
</tr>
<tr>
<td>192</td>
<td>144</td>
<td>1920</td>
<td>13x smaller</td>
<td>65.8</td>
</tr>
<tr>
<td>256</td>
<td>198</td>
<td>3840</td>
<td>19x smaller</td>
<td>285.0</td>
</tr>
</tbody>
</table>

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

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

René Struik, e-mail: rstruik.ext@gmail.com
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.

Conversion of ECDSA to Fast Verify friendly format: via simple post-processing
“Friendly ECDSA” 😊

René Struik, e-mail: rstruik.ext@gmail.com
Friendly ECDSA scheme

**System-wide parameters**
Elliptic curve of prime order \( n \) with generator \( G \). Hash function \( h \).

**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 \( s := k^{-1}(e + d \cdot r) \mod n \).
5. If \( r \notin [1, n-1] \) or \( s \notin [1, n-1] \), go to #2.
6. Return \((r, s)\) if \( y\)-coordinate of \( R \) even; return \((r, -s)\) otherwise.

**Initial set-up**
Signer A selects private key \( d \in [1, n-1] \) and publishes its public key \( Q = dG \).

**Signature verification**

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

**OUTPUT:** Accept or reject signature.

**ACTIONS:**
1. If \( r \notin [1, n-1] \), return ‘reject’.
2. Map \( r \) to \( R \) (only one of \( R \) or \( -R \) valid, since \( y\)-coordinate of \( R \) or \( -R \) odd).
3. If \( s \notin [1, n-1] \), return ‘reject’.
4. Compute \( e := h(m) \).
5. Check that \( R = s^{-1}(e \cdot G + r \cdot Q) \).
   If verification succeeds, return ‘accept’; otherwise return ‘reject’.

Anyone can do this post-processing

Anyone can do this pre-processing

René Struik, e-mail: rstruik.ext@gmail.com
How to get to Friendly ECDSA

Existing ECDSA signatures
– Anyone can post-process legacy ECDSA certificate and put into friendly format
– This could be device that participates into key agreement (ECDH + signed exponents)

New ECDSA signatures
– Generate in friendly format
– If verifier knows, he can always get speed-ups
  explicit method: use, e.g., new OID with PKIX, etc.
  implicit method: facilitate in IETF drafts currently in pipeline (no need for new OIDs)
– If verifier does not know, he can guess (best: +40%, worst: -12%, avg.: +8%)

Note:
– Devices that do not implement speed-ups will not notice, since compatible format
– Possible to move towards implementing verification speed-ups over time (one can change one’s mind)

René Struik, e-mail: rstruik.ext@gmail.com
IPR – where is it?

Potential IPR strings attached to following techniques:

− Accelerated verification of ECDSA signatures
− Combined ECDSA signature verification and ECC-based key agreement (e.g., ECDH with ECDSA signed exponents)

Ref: [https://datatracker.ietf.org/ipr/1363/](https://datatracker.ietf.org/ipr/1363/)

Hence, making techniques *optional* to use for those who choose to do so

**Ideal scenario:**
Everyone facilitates others to fully benefit from speed-ups should they choose to do so.
Further reading


7. R. Struik, ‘Batch Computations Revisited: Combining Key Computations and Batch Verifications,’ to be presented at SAC 2010, Waterloo, ON, Canada, August 12-13, 2010.

René Struik, e-mail: rstruij.ext@gmail.com
Back-up slides with more technical detail

René Struijk
e-mail: rstruij.ext@gmail.com

Results based on work conducted at Certicom Research
Part I –
Accelerated Verification of ECDSA Signatures

René Struik
e-mail: rstruik.ext@gmail.com

Joint work with A. Antipa, D.R. Brown, R. Gallant, R. Lambert, S.A. Vanstone
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
• Conclusions
### ECDSA signature scheme

**System-wide parameters**

- Elliptic curve of prime order \(n\) with generator \(G\). Hash function \(h\).

**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 \(s := k^{-1}(e + dr) \mod n\).
5. If \(r, s \in [1, n-1]\), return \((r, s)\); otherwise, go to Step 2.

**Initial set-up**

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

**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.

**René Struik, e-mail:** rstruik.ext@gmail.com
**Fast ECDSA signature scheme**

<table>
<thead>
<tr>
<th>System-wide parameters</th>
<th>Initial set-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic curve of prime order $n$ with generator $G$. Hash function $h$.</td>
<td>Signer A selects private key $d \in [1,n-1]$ and publishes its public key $Q = dG$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signature generation</th>
<th>Signature verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT: Message $m$, private key $d$. OUTPUT: Signature $(R, s)$.</td>
<td>INPUT: Message $m$, signature $(R, s)$; Public signing key $Q$ of Alice. OUTPUT: Accept or reject signature.</td>
</tr>
</tbody>
</table>

**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 $s := k^{-1}(e + dr) \mod n$.  
5. If $r, s \in [1,n-1]$, return $(R, s)$; otherwise, go to Step 2.

**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 $R = s^{-1}(eG + rQ)$. 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.

**René Struik, e-mail:** rstruik.ext@gmail.com
### Fast ECDSA signature scheme

#### Computational aspects

<table>
<thead>
<tr>
<th>Ordinary signature verification</th>
<th>Fast signature verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONS:</td>
<td>ACTIONS:</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3. Compute $R' := (e \cdot s^{-1}) \cdot G + (r \cdot s^{-1}) \cdot Q$.</td>
<td>2. Map $R$ to $r$.</td>
</tr>
<tr>
<td>4. Check that $R'$ maps to $r$.</td>
<td>4. Check that $R = (e \cdot s^{-1}) \cdot G + (r \cdot s^{-1}) \cdot Q$.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Ordinary signature verification**

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

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

**Fast signature verification**

Evaluate expression $\Delta := s^{-1} \cdot (e \cdot G + r \cdot Q) - R$ 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 … ☹️

*René Struik, e-mail: rstruik.ext@gmail.com*
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 $\lambda$ and $\mu$ 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.

René Struik, e-mail: rstruik.ext@gmail.com
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 \( \lambda \) and \( \mu \) 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 \( \lambda \) and \( \mu \) have size half the bit-length of \( n \).
Write \( \mu e s^{-1} \equiv \alpha_0 + \alpha_1 t \pmod{n} \), where \( \alpha_0, \alpha_1 \) have size half the bit-length of \( n \).
Evaluate \( \mu \Delta := (\mu e s^{-1}) G + \lambda Q - \mu R \)
\[ = \alpha_0 G_0 + \alpha_1 G_1 + \lambda Q - \mu R \]

Cost: half-size combination of known points \( G_0, G_1 \) and unknown points \( Q, R \).

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

Cost: full-size linear combination of known point \( G \) and unknown point \( Q \).

René Struik, e-mail: rstruik.ext@gmail.com
Computational aspects (3)

Optimum choice
Write $r / s \equiv \lambda / \mu \pmod{n}$, where $\lambda$ and $\mu$ 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 $a \cdot x + n \cdot y = d$.

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 \mod u, u); X \leftarrow \begin{pmatrix} -q & 1 \\ 1 & 0 \end{pmatrix}X 
   } 
3. $(d, x, y) \leftarrow (v, x_{21}, x_{22})$.

Invariant:
\[ a \cdot x_{11} + n \cdot x_{12} = u \]
\[ a \cdot x_{21} + n \cdot x_{22} = v \]

Let $a := r \cdot s^{-1} \pmod{n}$.
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 $\mu := x_{11}$.

René Struik, e-mail: rstruik.ext@gmail.com
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

<table>
<thead>
<tr>
<th>P-384 curve</th>
<th>ECDSA Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC operations</td>
<td>Ordinary</td>
</tr>
<tr>
<td>– Add</td>
<td>194</td>
</tr>
<tr>
<td>– Double</td>
<td>384</td>
</tr>
<tr>
<td>– Total(^1)</td>
<td>459</td>
</tr>
</tbody>
</table>

\(^1\)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

René Struik, e-mail: rstrui.ekt@gmail.com
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.

René Struik, e-mail: rstruik.ext@gmail.com
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)

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>Verification cost (ms)</th>
<th>Ratio fast ECDSA verify vs. RSA verify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RSA\textsuperscript{2}</td>
<td>ECDSA\textsuperscript{2}</td>
</tr>
<tr>
<td>80</td>
<td>1.4</td>
<td>4.0</td>
</tr>
<tr>
<td>112</td>
<td>5.2</td>
<td>7.7</td>
</tr>
<tr>
<td>128</td>
<td>11.0</td>
<td>11.8</td>
</tr>
<tr>
<td>192</td>
<td>65.8</td>
<td>32.9</td>
</tr>
<tr>
<td>256</td>
<td>285.0</td>
<td>73.2</td>
</tr>
</tbody>
</table>

\textsuperscript{1}Excluding (fixed) overhead of identification data
\textsuperscript{2}Certicom Security Builder
\textsuperscript{3}Estimate

Conclusion
Efficiency advantage of RSA signatures over ECDSA signatures is vanishing

René Struik, e-mail: rstruik.ext@gmail.com
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

**René Struik, e-mail:** rstruikeext@gmail.com
Part II –
Combined Verification and Key Computation

René Struik
e-mail: rstruik.ext@gmail.com
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
• Conclusions
Public key cryptography

Communication model
Communicating parties a priori share authentic information

René Struik, e-mail: rstruik.ext@gmail.com
Anonymous Diffie-Hellman (ephemeral ECDH)

ACTIONS:
1. Select $a \in_R [1, n-1]$. 

Bob 

Random $A = aG$

Alice 

Random $B = bG$

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

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.

René Struik, e-mail: rstruij.ext@gmail.com
Key agreement schemes

Authenticated Diffie-Hellman (static ECDH)

**ACTIONS:**
1. Verify $\text{Cert}_{CA}(Bob, B)$.

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

**Properties**
- **Key agreement:** Both parties arrive at the 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.

René Struik, e-mail: rstruik.ext@gmail.com
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.

René Struik, e-mail: rstruik.ext@gmail.com
Key agreement schemes

Computational aspects

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.

René Struik, e-mail: rstruik.ext@gmail.com
ECDSA signature scheme

<table>
<thead>
<tr>
<th>ECDSA signature verification</th>
<th>System-wide parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT:</strong> Message $m$, signature $(r, s)$;</td>
<td>Elliptic curve with generator $G$.</td>
</tr>
<tr>
<td>Public signing key $Q$ of Alice.</td>
<td>Hash function $h$.</td>
</tr>
<tr>
<td><strong>OUTPUT:</strong> Accept or reject signature.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ordinary signature verification</th>
<th>Fast signature verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIONS:</td>
<td>ACTIONS:</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1. Compute $e := h(m)$.</td>
<td>1. Compute $e := h(m)$.</td>
</tr>
<tr>
<td>2. Compute $R' := (e s^{-1}) G + (r s^{-1}) Q$.</td>
<td>2. Reconstruct $R$ from $r$.</td>
</tr>
<tr>
<td>3. Check that $R'$ maps to $r$.</td>
<td>3. Check that $R = (e s^{-1}) G + (r s^{-1}) Q$.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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.

René Struik, e-mail: rstruik.ext@gmail.com
Computational aspects (1)

Step 2: ECDH key computation (key establishment)

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 \[ s^{-1} (e \cdot G + r \cdot Q) - R = O, \]

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!

René Struik, e-mail: rstruik.ext@gmail.com

41
Computational aspects (2)

Step 2: ECDH key computation (key establishment)
Compute expression \( K := aB \).

Step 3: ECDSA certificate verification (key authentication)
Evaluate expression \( \Delta := s^{-1}(eG + rQ) - R \) and check that \( \Delta = O \).

Step 2 and Step 3 combined: Combined verification and key computation
Compute expression \( K' := aB + \lambda(s^{-1}(eG + rQ) - R) \) instead.

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

René Struik, e-mail: rstruik.ext@gmail.com
Computational aspects (3)

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

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

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’.

René Struik, e-mail: rstruik.ext@gmail.com
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 \( s^{-1} (eG + rQ) = R \).
Cost: linear combination of known point \( G \) and unknown point \( Q \).

Step 2 and Step 3 combined: Combined verification and key computation
Compute expression \( K' := aB - \lambda R + (\lambda es^{-1})G + (\lambda rs^{-1})Q \).
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.

René Struik, e-mail: rstruik.ext@gmail.com
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

<table>
<thead>
<tr>
<th>P-384 curve</th>
<th>ECDH key</th>
<th>ECDSA (incremental cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC operations</td>
<td></td>
<td>Separately</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ordinary</td>
</tr>
<tr>
<td>Add</td>
<td>128</td>
<td>194</td>
</tr>
<tr>
<td>Double</td>
<td>384</td>
<td>384</td>
</tr>
<tr>
<td>Total¹</td>
<td>393</td>
<td>459</td>
</tr>
</tbody>
</table>

¹Normalized (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

<table>
<thead>
<tr>
<th>P-384 curve</th>
<th>ECMQV key</th>
<th>ECDSA (incremental cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC operations</td>
<td></td>
<td>Separately</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ordinary</td>
</tr>
<tr>
<td>– Add</td>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>– Double</td>
<td>384</td>
<td>384</td>
</tr>
<tr>
<td>– Total¹</td>
<td>459</td>
<td>459</td>
</tr>
</tbody>
</table>

¹Normalized (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)

René Struik, e-mail: rstruik.ext@gmail.com
Example (3)

Static ECDH and ECMQV with ECDSA certificates

<table>
<thead>
<tr>
<th>P-384 curve</th>
<th>Key computation</th>
<th>Key computation + ECDSA (total cost)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total ECC</td>
<td></td>
<td>ECDSA separately</td>
</tr>
<tr>
<td>operations$^1$</td>
<td></td>
<td>Ordinary</td>
</tr>
<tr>
<td>ECDH</td>
<td>393</td>
<td>852</td>
</tr>
<tr>
<td>ECMQV</td>
<td>459</td>
<td>918</td>
</tr>
</tbody>
</table>

$^1$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)

René Struik, e-mail: rstrui.k.ext@gmail.com
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)

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>Certificate size¹ (bytes)</th>
<th>Ratio ECC/RSA certificates</th>
<th>Verify – incremental cost (ms)</th>
<th>Ratio ECDSA verify vs. RSA verify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ECDSA</td>
<td>RSA</td>
<td></td>
<td>RSA²</td>
</tr>
<tr>
<td>80</td>
<td>72</td>
<td>256</td>
<td>4x smaller</td>
<td>1.4</td>
</tr>
<tr>
<td>112</td>
<td>84</td>
<td>512</td>
<td>6x smaller</td>
<td>5.2</td>
</tr>
<tr>
<td>128</td>
<td>96</td>
<td>768</td>
<td>8x smaller</td>
<td>11.0</td>
</tr>
<tr>
<td>192</td>
<td>144</td>
<td>1920</td>
<td>13x smaller</td>
<td>65.8</td>
</tr>
<tr>
<td>256</td>
<td>198</td>
<td>3840</td>
<td>19x smaller</td>
<td>285.0</td>
</tr>
</tbody>
</table>

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

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

René Struik, e-mail: rstruik.ext@gmail.com
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