Signatures
and DLP-I

Tanja Lange
Technische Universiteit Eindhoven
How to compute $aP$

Use binary representation of $a$ to compute $a(X, Y)$ in $\lceil \log_2 a \rceil$ doublings and at most that many additions.

E.g. $a = 23 = (10111)_2$:

For $a = (1, a_{n-1}, \ldots, a_1, a_0)_2$; compute scalar multiplication
$aP = 2(\cdots 2(2(2P + a_{n-1}P) + a_{n-2}P) + \cdots + a_1P) + a_0P$.

There are many more efficient methods.
ECDSA

Users can sign messages using Edwards curves.

Take a point $P$ on an Edwards curve modulo a prime $q > 2$.

ECDSA signer needs to know the order of $P$.

There are only finitely many other points; about $q$ in total.

Adding $P$ to itself will eventually reach $(0, 1)$; let $\ell$ be the smallest integer $> 0$ with $\ell P = (0, 1)$.

This $\ell$ is the order of $P$. 
The signature scheme has as system parameters a curve $E$; a base point $P$; and a hash function $h$ with output length at least $\lceil \log_2 \ell \rceil + 1$.

Alice’s secret key is an integer $a$ and her public key is $P_A = aP$.

To sign message $m$, Alice computes $h(m)$; picks random $k$; computes $R = kP = (x_1, y_1)$; puts $r \equiv y_1 \mod \ell$; computes $s \equiv k^{-1}(h(m) + r \cdot a) \mod \ell$.

The signature on $m$ is $(r, s)$. 
Anybody can verify signature given \( m \) and \((r, s)\):
Compute \( w_1 \equiv s^{-1}h(m) \mod l \) and \( w_2 \equiv s^{-1} \cdot r \mod l \).
Check whether the \(y\)-coordinate of \( w_1P + w_2P_A \) equals \( r \) modulo \( l \) and if so, accept signature.

Alice’s signatures are valid:
\[
\begin{align*}
w_1P + w_2P_A &= (s^{-1}h(m))P + (s^{-1} \cdot r)P_A \\
&= (s^{-1}(h(m) + rA))P = kP
\end{align*}
\]
and so the \(y\)-coordinate of this expression equals \( r \),
the \(y\)-coordinate of \( kP \).
Attacker’s view on signatures

Anybody can produce an $R = kP$. Alice’s private key is only used in
$s \equiv k^{-1}(h(m) + r \cdot a) \mod \ell$.

Can fake signatures if one can break the DLP, i.e., if one can compute $a$ from $P_A$.
Most of my lectures deal with methods for breaking DLPs.
Sometimes attacks are easier...
If $k$ is known for some $m$, $(r, s)$ then $a \equiv (sk - h(m))/r \mod \ell$.

If two signatures $m_1, (r, s_1)$ and $m_2, (r, s_2)$ have the same value for $r$: assume $k_1 = k_2$; observe $s_1 - s_2 = k_1^{-1}(h(m_1) + ra - (h(m_2) + ra))$; compute $k = (s_1 - s_2)/(h(m_1) - h(m_2))$. Continue as above.

If bits of many $k$’s are known (biased PRNG) can attack $s \equiv k^{-1}(h(m) + r \cdot a) \mod \ell$ as hidden number problem using lattice basis reduction.
Malicious signer

Alice can set up her public key so that two messages of her choice share the same signature, i.e., she can claim to have signed \( m_1 \) or \( m_2 \) at will:

\[
R = (x_1, y_1) \quad \text{and} \quad -R = (-x_1, y_1)
\]
have the same \( y \)-coordinate.

Thus, \((r, s)\) fits

\[
R = kP, \quad s \equiv k^{-1}(h(m_1) + ra) \mod \ell \quad \text{and} \quad -R = (-k)P, \quad s \equiv -k^{-1}(h(m_2) + ra) \mod \ell
\]

if

\[
a \equiv -(h(m_1) + h(m_2))/2r \mod \ell.
\]
Malicious signer

Alice can set up her public key so that two messages of her choice share the same signature, i.e., she can claim to have signed $m_1$ or $m_2$ at will:

$$R = (x_1, y_1) \text{ and } -R = (-x_1, y_1)$$

have the same $y$-coordinate.

Thus, $(r, s)$ fits $R = kP$, $s \equiv k^{-1}(h(m_1) + ra) \mod \ell$ and $-R = (-k)P$, $s \equiv -k^{-1}(h(m_2) + ra) \mod \ell$ if $a \equiv -(h(m_1) + h(m_2))/2r \mod \ell$.

(Easy tweak: include bit of $x_1$.)
EdDSA


Uses $k = \text{hash}(b, m)$; $b$ is second secret key.

Make $h$ dependent on $R$ and $P_A$: 
$h = \text{hash}(R, P_A, m)$.

No inversions mod $\ell$: 
$s \equiv k + ha \mod \ell$.

Verification:

does $sP$ equal $R + hP_A$?

Use Edwards curves!

Very fast signing and verifying.
Edwards curves are cool
Elliptic-curve groups

\[ y^2 = x^3 + ax + b. \]
Elliptic-curve groups

\[ y^2 = x^3 + ax + b. \]
Elliptic-curve groups

$y^2 = x^3 + ax + b.$
Also neutral element at $\infty$.
$-(x, y) = (x, -y)$. 
\((x_W, y_W) + (x_R, y_R) = (x_{W+R}, y_{W+R}) = (\lambda^2 - x_W - x_R, \lambda(x_W - x_{W+R}) - y_W)\).

\(x_W \neq x_R\), “addition”:
\[\lambda = (y_R - y_W)/(x_R - x_W).\]
Total cost 1I + 2M + 1S.

\(W = R\) and \(y_W \neq 0\), “doubling”:
\[\lambda = (3x_W^2 + a)/(2y_W).\]
Total cost 1I + 2M + 2S.

Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader.
\((x_W, y_W) + (x_R, y_R) = (x_{W+R}, y_{W+R}) = (\lambda^2 - x_W - x_R, \lambda(x_W - x_{W+R}) - y_W).\)

\(x_W \neq x_R, \text{ “addition”:} \lambda = (y_R - y_W)/(x_R - x_W).\)

Total cost 1I + 2M + 1S.

\(W = R \text{ and } y_W \neq 0, \text{ “doubling”:} \lambda = (3x_W^2 + a)/(2y_W).\)

Total cost 1I + 2M + 2S.

Following algorithms will need a unique representative per point. For that Weierstrass curves are the speed leader . . . and I thought turtles were defensive.
The discrete-logarithm problem

Define \( p = 1000003 \) and consider the Weierstrass curve
\[ y^2 = x^3 - x \] over \( \mathbb{F}_p \).
This curve has \( 1000004 = 2^2 \cdot 53^2 \cdot 89 \) points and \( P = (101384, 614510) \) is a point of order \( 2 \cdot 53^2 \cdot 89 \).
In general, point counting over \( \mathbb{F}_p \) runs in time polynomial in \( \log p \).
Number of points in
\[ [p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}] \].
The group is isomorphic to \( \mathbb{Z}/n \times \mathbb{Z}/m \), where \( n \mid m \) and \( n \mid (p - 1) \).
Can we find an integer \( n \in \{1, 2, 3, \ldots, 500001\} \) such that \( nP = (670366, 740819) \)?

This point was generated as a multiple of \( P \); could also be outside cyclic group.

Could find \( n \) by brute force. Is there a faster way?
Understanding brute force

Can compute successively

\(1P = (101384, 614510),\)
\(2P = (102361, 628914),\)
\(3P = (77571, 87643),\)
\(4P = (650289, 31313),\)
\(500001P = -P.\)
\(500002P = \infty.\)

At some point we’ll find \(n\)
with \(nP = (670366, 740819).\)

Maximum cost of computation:
\(\leq 500001\) additions of \(P;\)
\(\leq 500001\) nanoseconds on a CPU
that does 1 ADD/nanosecond.
This is negligible work for $p \approx 2^{20}$.

But users can standardize a larger $p$, making the attack slower.

Attack cost scales linearly:
$\approx 2^{50}$ ADDs for $p \approx 2^{50}$,
$\approx 2^{100}$ ADDs for $p \approx 2^{100}$, etc.

(Not exactly linearly: cost of ADDs grows with $p$.
But this is a minor effect.)
Computation has a good chance of finishing earlier.
Chance scales linearly:
1/2 chance of 1/2 cost;
1/10 chance of 1/10 cost; etc.

“So users should choose large \( n \).”

That’s pointless. We can apply “random self-reduction”:
choose random \( r \), say 69961;
compute \( r P = (593450, 987590) \);
compute \( (r + n)P \) as \( (593450, 987590) + (670366, 740819) \);
compute discrete log;
subtract \( r \mod 500002 \); obtain \( n \).
Computation can be parallelized.

One low-cost chip can run many parallel searches. Example, $2^6 \in$: one chip, $2^{10}$ cores on the chip, each $2^{30}$ ADDs/second? Maybe; see SHARCS workshops for detailed cost analyses.

Attacker can run many parallel chips. Example, $2^{30} \in$: $2^{24}$ chips, so $2^{34}$ cores, so $2^{64}$ ADDs/second, so $2^{89}$ ADDs/year.
Multiple targets and giant steps

Computation can be applied to many targets at once.

Given 100 DL targets $n_1P$, $n_2P$, $\ldots$, $n_{100}P$:
Can find all of $n_1$, $n_2$, $\ldots$, $n_{100}$ with $\leq 500002$ ADDs.

Simplest approach: First build a sorted table containing $n_1P$, $\ldots$, $n_{100}P$.
Then check table for $1P$, $2P$, etc.
Interesting consequence #1: Solving all 100 DL problems isn’t much harder than solving one DL problem.

Interesting consequence #2: Solving at least one out of 100 DL problems is much easier than solving one DL problem.

When did this computation find its first $n_i$?

Typically $\approx 500002/100$ mults.
Can use random self-reduction to turn a single target into multiple targets.

Given $nP$:
Choose random $r_1, r_2, \ldots, r_{100}$.
Compute $r_1 P + nP$, $r_2 P + nP$, etc.

Solve these 100 DL problems.
Typically $\approx \ell/100$ mults to find at least one $r_i + n \mod \ell$, immediately revealing $n$. 
Also spent some ADDs to compute each $r_i P$:
$\approx \lg p$ ADDs for each $i$.
Faster: Choose $r_i = ir_1$
with $r_1 \approx \ell/100$.
Compute $r_1 P$;
$r_1 P + n P$;
$2r_1 P + n P$;
$3r_1 P + n P$; etc.
Just 1 ADD for each new $i$.
$\approx 100 + \lg \ell + \ell/100$ ADDs
to find $n$ given $nP$.  

Faster: Increase 100 to $\approx \sqrt{\ell}$.
Only $\approx 2\sqrt{\ell}$ ADDs

to solve one DL problem!

“Shanks baby-step-giant-step
discrete-logarithm algorithm.”

Example: $p = 1000003$, $\ell = 500002$, $P = (101384, 614510)$,

$Q = nP = (670366, 740819)$.

Compute $708P = (393230, 421116)$.

Then compute 707 targets:

$708P + Q = (342867, 153817)$,

$2 \cdot 708P + nP = (430321, 994742)$,

$3 \cdot 708P + nP = (423151, 635197)$,

$\ldots$, $706 \cdot 708P + nP = (534170, 450849)$. 
Build a sorted table of targets:

$600 \cdot 708P + Q = (799978, 929249)$,
$219 \cdot 708P + Q = (425475, 793466)$,
$679 \cdot 708P + Q = (996985, 191440)$,
$242 \cdot 708P + Q = (262804, 347755)$,
$27 \cdot 708P + Q = (785344, 831127)$,

\[ \vdots \]

$317 \cdot 708P + Q = (599785, 189116)$. 

Look up $P, 2P, 3P$, etc. in table.

$620P = (950652, 688508)$; find
$596 \cdot 708P + Q = (950652, 688508)$
in the table of targets;
so $620 = 596 \cdot 708P + n \mod 500002$; deduce $n = 78654$. 
Factors of the group order

$P$ has order $2 \cdot 53^2 \cdot 89$.

Given $Q = nP$, find $n = \log_P Q$:

$R = (53^2 \cdot 89)P$ has order 2, and $S = (53^2 \cdot 89)Q$ is multiple of $R$.
Compute $n_1 = \log_R S \equiv n \mod 2$.

$R = (2 \cdot 53 \cdot 89)P$ has order 53, and
$S = (2 \cdot 53 \cdot 89)Q$ is multiple of $R$.
Compute $n_2 = \log_R S \equiv n \mod 53$.

This is a DLP in a group of size 53.
\[ T = (2 \cdot 89)(Q - n_2P) \] is also a multiple of \( R \).

Compute \( n_3 = \log_R T \equiv n \mod 53 \).

Now \( n_2 + 53n_3 \equiv n \mod 53^2 \).

\( R = (2 \cdot 53^2)P \) has order 89, and \( S = (2 \cdot 53^2)Q \) is multiple of \( R \).

Compute \( n_4 = \log_R S \equiv n \mod 89 \).

Use Chinese Remainder Theorem

\[
\begin{align*}
n &\equiv n_1 \mod 2, \\
n &\equiv n_2 + 53n_3 \mod 53^2, \\
n &\equiv n_4 \mod 89,
\end{align*}
\]

to determine \( n \) modulo \( 2 \cdot 53^2 \cdot 89 \).
This “Pohlig-Hellman method” converts an order-\(ab\) DL into an order-\(a\) DL, an order-\(b\) DL, and a few scalar multiplications.

Here \((53^2 \cdot 89)P = (1, 0)\) and \((53^2 \cdot 89)Q = \infty\), thus \(n_1 = 0\).

\((2 \cdot 53 \cdot 89)P = (539296, 488875)\),
\((2 \cdot 53 \cdot 89)Q = (782288, 572333)\).

A search quickly finds \(n_2 = 2\).
\((2 \cdot 89)(Q - 2P) = \infty\), thus \(n_3 = 0\) and \(n_2 + 53n_3 = 2\).
\[(2 \cdot 53^2)P = (877560, 947848) \text{ and } (2 \cdot 53^2)Q = (822491, 118220)\].

Compute \(n_4 = 67\), e.g. using BSGS.

Use Chinese Remainder Theorem

\[n \equiv 0 \mod 2,\]
\[n \equiv 2 \mod 53^2,\]
\[n \equiv 67 \mod 89,\]

to determine \(n = 78654\).

Pohlig-Hellman method reduces security of discrete logarithm problem in group generated by \(P\) to security of largest prime order subgroup.
The rho method

Simplified, non-parallel rho:
Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

Birthday paradox:
Randomly choosing from $\ell$
elements picks one element twice
after about $\sqrt{\pi \ell / 2}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm
(e.g., Floyd) quickly detects this.
Assume that for each point we know $a_i, b_i \in \mathbb{Z}/\ell\mathbb{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$.

If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i) / (b_i - b_j)$. 
Assume that for each point we know $a_i, b_i \in \mathbb{Z}/\ell\mathbb{Z}$ so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that $a_i P + b_i Q = a_j P + b_j Q$ so $(b_i - b_j)Q = (a_j - a_i)P$.

If $b_i \neq b_j$ the DLP is solved: $n = (a_j - a_i)/(b_i - b_j)$.

e.g. $f(W_i) = a(W_i)P + b(W_i)Q$, starting from some initial combination $W_0 = a_0 P + b_0 Q$.

If any $W_i$ and $W_j$ collide then $W_{i+1} = W_{j+1}, W_{i+2} = W_{j+2}$, etc.
If functions $a(W)$ and $b(W)$ are random modulo $\ell$, iterations perform a random walk in $\langle P \rangle$. If $a$ and $b$ are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on equivalence classes under $\pm$.

There are only $[\ell/2]$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard’s rho method can be combined with any easily computed group automorphism of small order. More on that later.
Parallel collision search

Running Pollard’s rho method on $N$ computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

Want better way to spread computation across clients. Want to find collisions between walks on different machines, without frequent synchronization!

Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be distinguished points.
Parallel rho: Perform many walks with different starting points but same update function $f$. If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a distinguished point and report the point along with $a_i$ and $b_i$ to server.

Server receives, stores, and sorts all distinguished points. Two walks reaching same distinguished point give collision. This collision solves the DLP.
Attacker chooses frequency and definition of distinguished points. Tradeoffs are possible:
If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.
If distinguished points are frequent, many shorter walks will be performed.
In any case do not wait for cycle.
Total # of iterations unchanged.