Elliptic Curve Digital Signature Algorithm

Kevin Linger and Palmer Dabbelt

January 14, 2014

Public Key Cryptography

- Two keys (called a keypair)
  - Public: the whole world can know this
  - Private: only the owner knows this
- Four operations
  - `Sign(Message, Private)` $\rightarrow$ `Signature`
  - `Verify(Message, Signature, Public)` $\rightarrow$ `Boolean`
  - `Encrypt(Message, Public)` $\rightarrow$ `Cyphertext`
  - `Decrypt(Cyphertext, Private)` $\rightarrow$ `Message`
- ECDSA is a widely-used (SSH, SSL/TLS) signature algorithm
  - Supports `Sign` and `Verify`

Cryptography and the Swarm

- Original motivation: the universal dataplane
  - Storage component of SwarmOS
  - Large network (10^{10} nodes)
  - Many low power sensors (μ**W**)
  - Some high power servers
- Goal: all storage is fault-tolerant
  - Byzantine fault tolerence algorithm
  - Sign every message on the sensors
- Problem: ECDSA is **mJ** per signature!

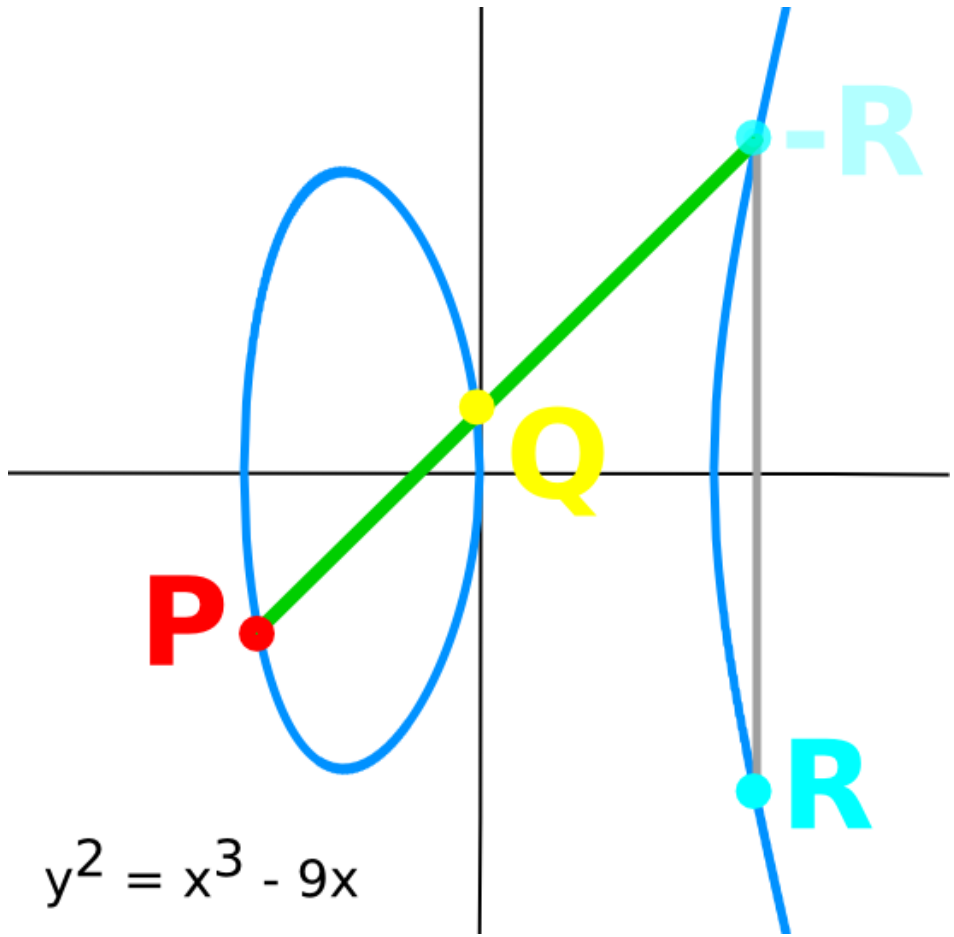
The Discrete Logarithm Problem

- $b^k = g$
  - Difficult in one direction (given b and g find k)
  - Easy in another direction (given b and k find g)
- At the center of many public-key crypto schemes
- Only asymetric for some groups
  - $(\mathbb{Z}_p)^\times$: Integers modulo some prime, repeating multiplication
  - $GF(p)$: Elliptic curves modulo some prime, repeating addition
- Integer fields have a sub-linear algroithm

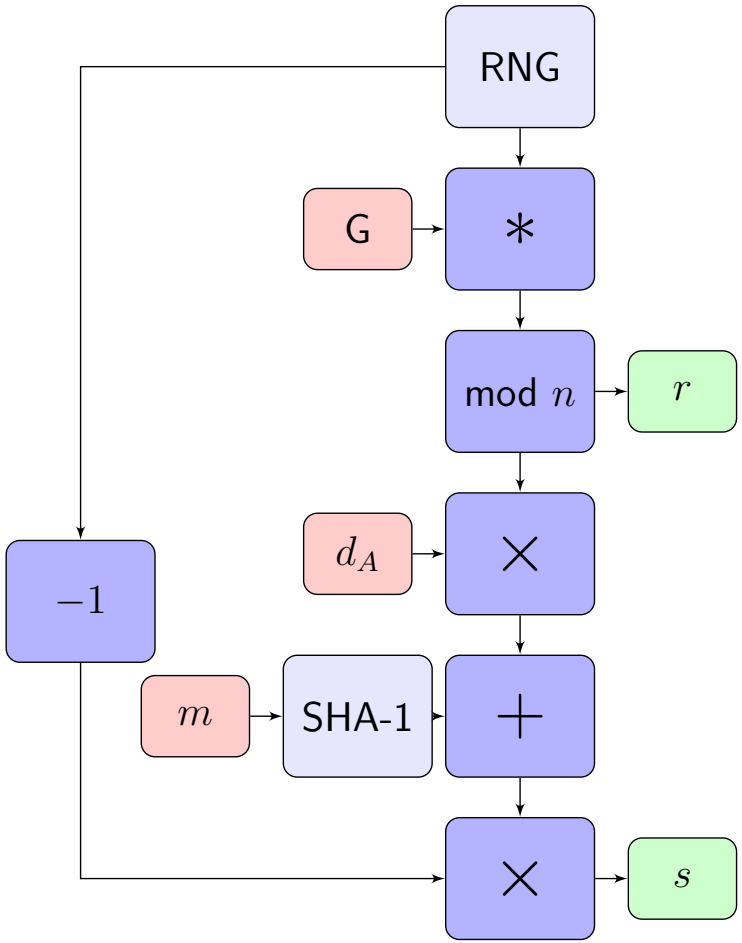
Security Level	DSA	ECDSA	Ratio
80	1024	160	3:1
112	2048	224	6:1
128	3072	256	10:1
192	7680	384	32:1
256	15360	512	64:1

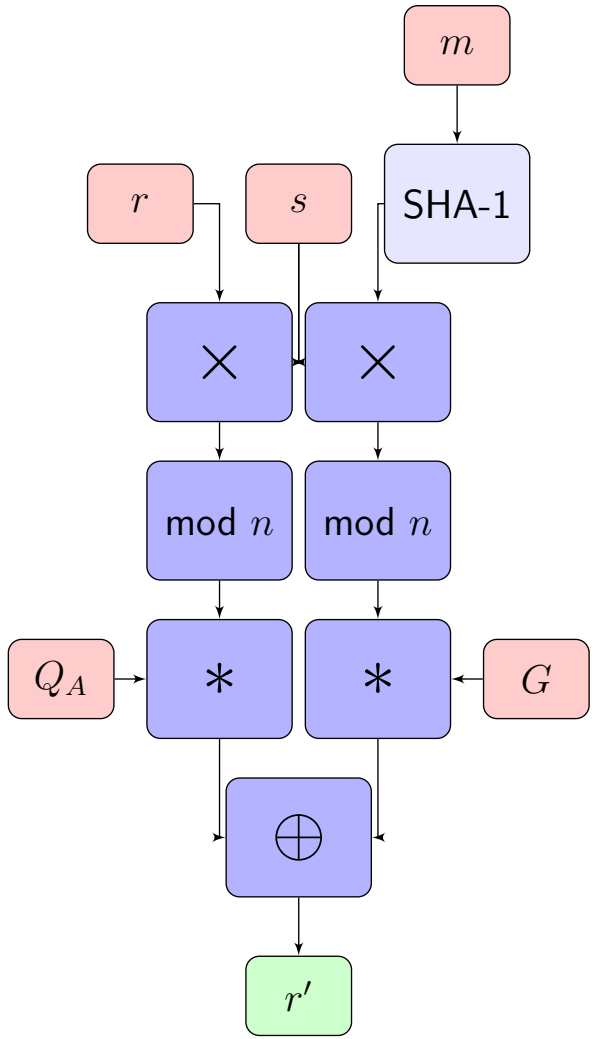
Elliptic Curve Cryptography

- Elliptic curve: the set of points that satisfy $y^2 = x^3 + ax + b$
  - Need a finite field $\rightarrow$ everything's modular
  - Needs to be difficult $\rightarrow$ modulo a large number
- Add and multiply defined to satsify a field



ECDSA





Point Operations

- ECDSA is essentially point multiply and some cleanup
  - Point Multiply: O(million) cycles
  - Cleanup: O(thousand) cycles
- Point multiplication is defined as repeated point doubling
- Point doubling is defined as

$$\left(\frac{(3P_x^2 + a)}{(2P_y)} - P_x\right)^2 * \left(\frac{(3P_x^2 + a)}{(2P_y)}\right) - P_y$$

- Simplifies to 5 modular multiplies and one modular divide
  - Modular operations are modulo a large (O(256-bit)) number

The Design Space

- A whole bunch of math
  - Data types: Points, Modular integers, and integers
  - Arithmatic operations: Add, Subtract, Multiply, Invert
- Everything boils down to integer arithmetic and control logic
  - What should be in software, what should be in hardware?
- Unfortunately, it's too big!
  - 2^{12} software configurations!

Hardware-Software Cotuning

- OpenSSL is the industry standard, but it's difficult to hack on
- Wrote our own ECDSA implementation in C++
- Generates software for a family of ECDSA accelerators

```
palmer palmer-caldesktop rocket-ecc $ time make check | tail -n0
real    12m1.390s
user    63m20.796s
sys     7m37.179s


palmer palmer-caldesktop rocket-ecc $ ptest | tail -n4
NRUN    6895
NPASS   6895
NFAIL   0
NEROR   0
```

The (Reduced) Design Space

Point Multiplication

- Control logic
- Point Addition
  - Control logic
  - Modular multiply
    - Control logic
    - Integer shifts, adds
  - Modular inverse
    - Control logic
    - Integer shifts, adds

- Two interesting hardware configurations
  - Point multiply hardware block
  - Modular multiply, inverse hardware blocks

Results

Platform	Power mJ/op	Speed op/sec	Area mm^2
OpenSSL (45nm)	20	1000	
x86 (45 nm)	4000	5	
Rocket	800	0.05	
Virtex 2 (90nm)	4000	250	
Virtex 6 (45nm)	500	4000	
Mod Mul	200	0.3	0.04
Mod Mul+Div	2	20	0.10
Point Add+Dbl	0.5	100	0.31
Point Mul	0.06	400	0.35

Future Work

- Time-space tradeoffs
  - Montgomery multipliers
  - Projective point representation
- Parallel software, multiple signatures in flight
- Hard-coded curve parameters
- 300MHz clock