High-speed cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

with some slides from:
Tanja Lange
Technische Universiteit Eindhoven

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow $\Rightarrow$ fewer users $\Rightarrow$ fewer cryptanalysts $\Rightarrow$ less attractive for everybody.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA verification computes $(S^{-1}H(M))B + (S^{-1}R)A$. OpenSSL has complicated code for fast computation of S^{-1}. Much simpler code would make verification considerably slower.

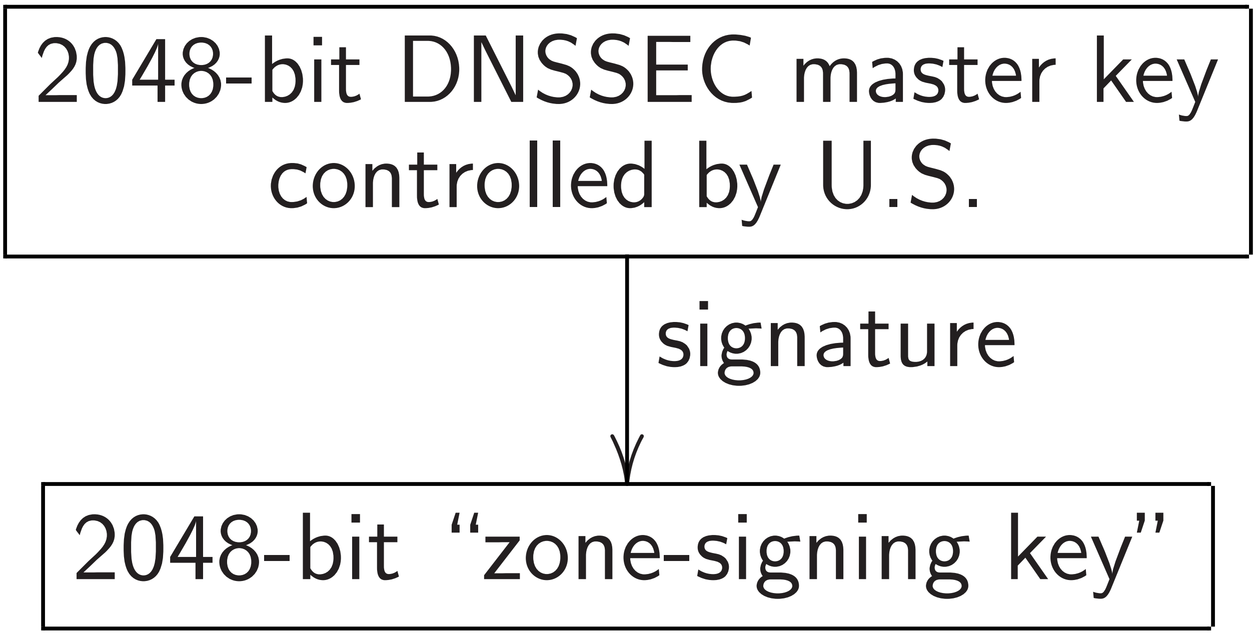
Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

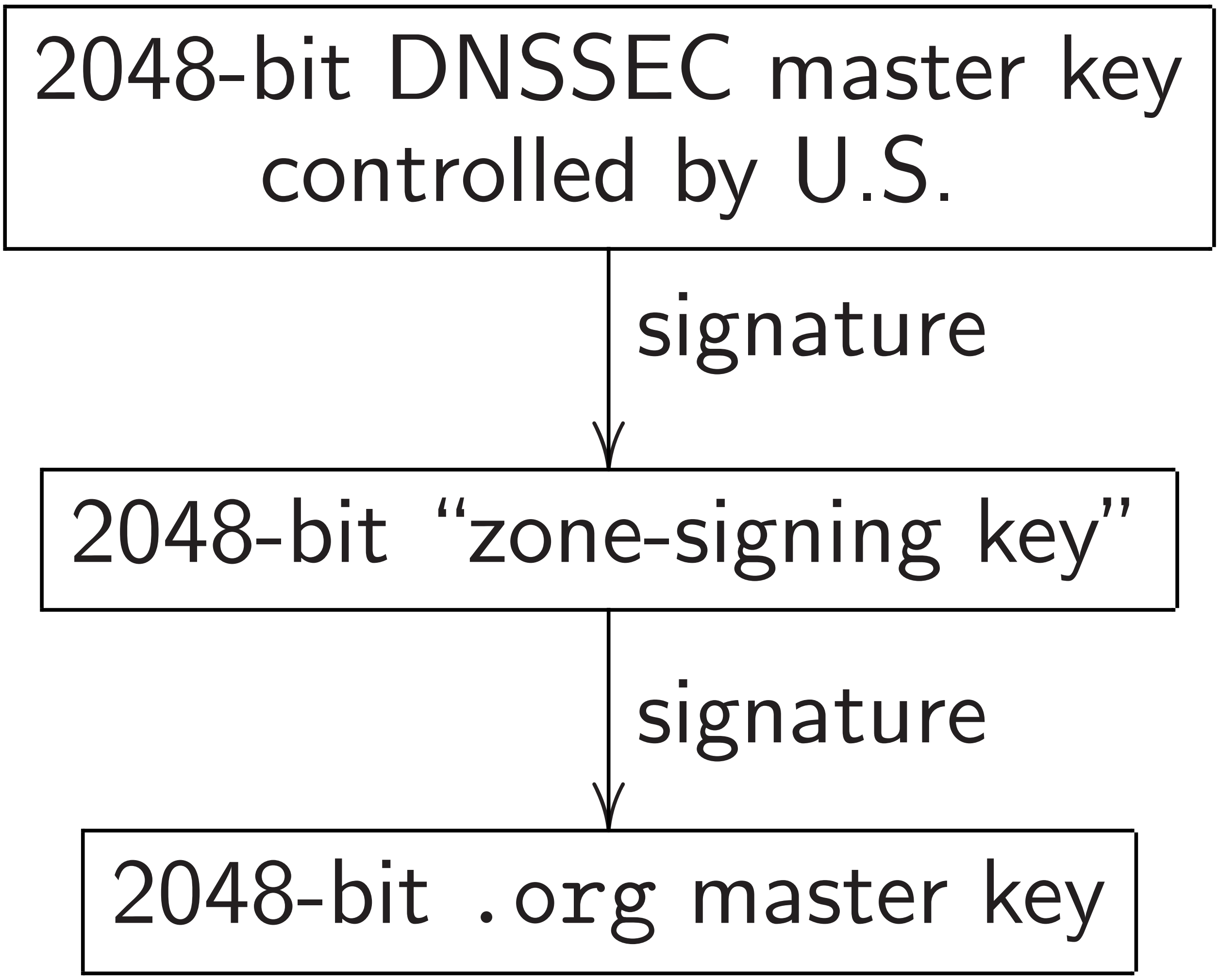
DNSSEC key sizes, 2016.11.28:

2048-bit DNSSEC master key controlled by U.S.

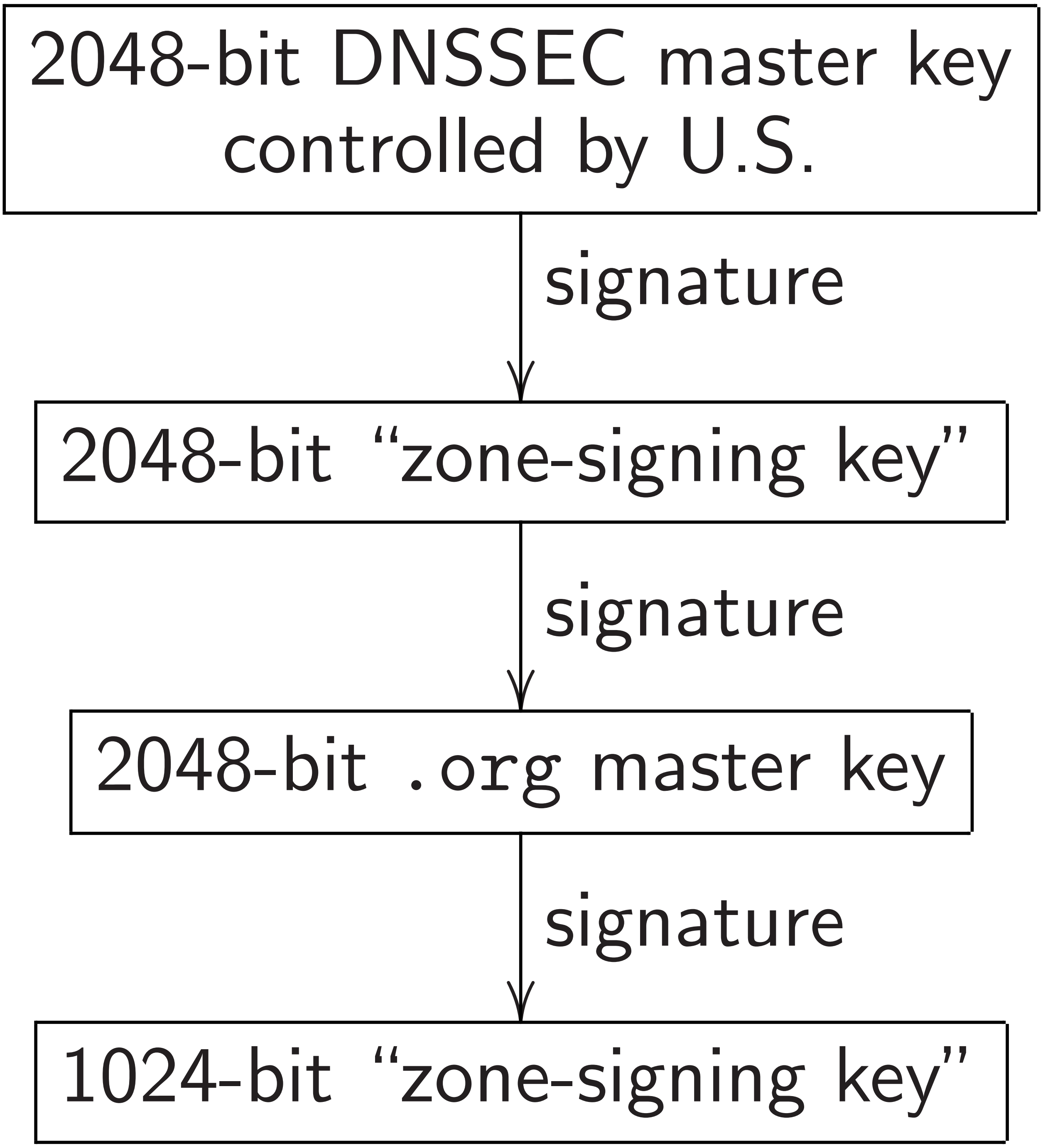
DNSSEC key sizes, 2016.11.28:



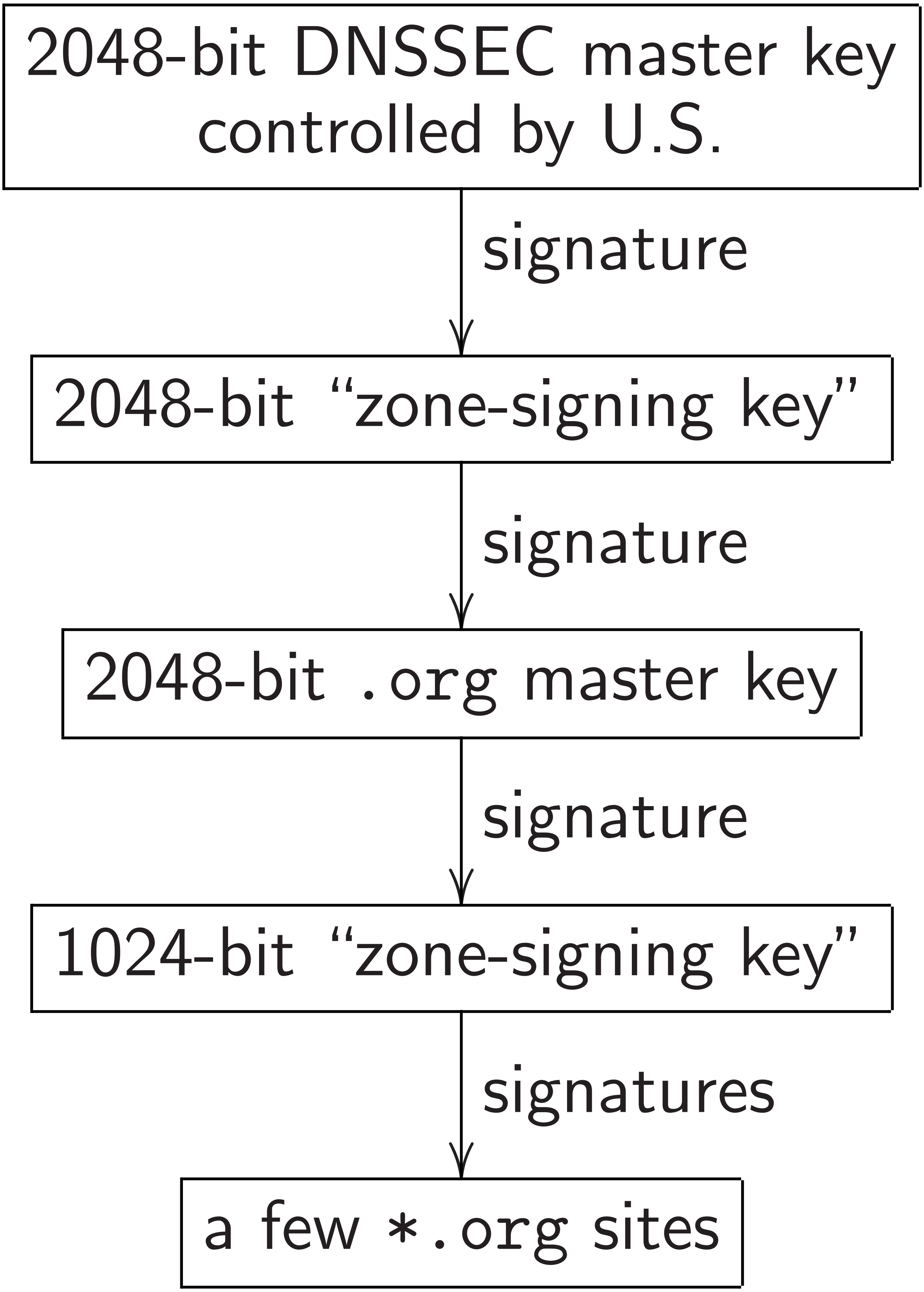
DNSSEC key sizes, 2016.11.28:



DNSSEC key sizes, 2016.11.28:



DNSSEC key sizes, 2016.11.28:



2011 Weimerskirch survey of security for car communication:

"V2V safety applications will broadcast 10 messages per second, and a vehicle will receive 1,000 or more messages per second. There are two approaches available to process such a high amount of messages: (1) only messages that might impose an actual impact to a vehicle are processed, or (2) dedicated security hardware to process all messages is applied."

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode . . . By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal.

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . .
Rijndael appears to offer an *adequate* security margin. . . .
Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

“The operations used by Serpent are among the easiest to defend against timing and power attacks.”

Hardware speed: “Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent’s speed is independent of key size.”

Great! Why didn’t Serpent win?

Aha: Software speed!

Aha: Software speed! “Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance.”

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

Conclusion: "NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism]."

Want fast *and* secure

Bad examples:
The pursuit of speed
damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Want fast *and* secure

Bad examples:
The pursuit of speed
damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed
without damaging security.

If security level was too low,
scale up: better security
for the same performance.

Success story: ECC.

Extensive work on speed of ECC at a high security level $\Rightarrow$ modern ECC is fast enough for practically all applications.

Requires serious analysis and optimization of algorithms. Not just "polynomial time"; not just "quadratic time".

Success story: ECC.

Extensive work on speed of ECC at a high security level $\Rightarrow$ modern ECC is fast enough for practically all applications.

Requires serious analysis and optimization of algorithms. Not just "polynomial time"; not just "quadratic time".

RSA and Rabin–Williams are even faster for signature verification, but slower for keygen, signing, sending keys, sending sigs.

Some signature-system history

1985 ElGamal: $\mathbf{F}_p^*$ signatures.

1990 Schnorr: ElGamal plus various improvements. Patented until 2008.

1991 DSA, announced by NIST, later credited to NSA: ElGamal with one Schnorr improvement.

1999 ECDSA: replacing $\mathbf{F}_p^*$ in DSA with an elliptic-curve group.

2011 EdDSA (e.g., Ed25519): Schnorr plus more improvements.

ElGamal verification:

(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod p$
and $R, S \in \{0, 1, \ldots, p - 2\}$.

Here p is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate random r,
compute $R = B^r \bmod p$,
$S = r^{-1}(H(M) - aR) \bmod p - 1$.

Hash the exponent

Tweak: (R, S) is signature of M if $B^{H(M)} \equiv A^{H(R)} R^S \pmod p$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Signer: as before except $S = r^{-1}(H(M) - aH(R)) \bmod p-1$.

Speed impact: negligible.
Hashing R is very fast.

Security impact: seems to be serious obstacle to any attack strategy that relies on choosing a particular A exponent.

Prime-order subgroup

Choose B to have order q for standard prime divisor q of $p-1$.
e.g. take 3000-bit p, 256-bit q.

Again verify $B^{H(M)} \equiv A^{H(R)} R^S$.
ECC: $H(M)B = H(R)A + SR$.

Signer: same except now
$S = r^{-1}(H(M) - aH(R)) \bmod q$.

Simpler security analysis.

Speed advantage: Smaller S (when q is smaller than $p-1$).
Less time to transmit signature.

Two scalars

Verify $B^{H(R)^{-1}H(M)} =$
$AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B =$
$A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^{S}$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A + \underline{S}R$.

Signer computes $\underline{S} =$
$r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A + \underline{S}R$.

Signer computes $\underline{S} = r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

From now on: Rename $\underline{S}$ as S.

Merge hashes: collision resilience

$B^{H(R,M)} = AR^S$.
ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$ is faster than $H(R)^{-1}H(M)$.

Security advantage: Imagine attacker somehow finding innocent M and dangerous M' with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if signer signs M then attacker reuses same signature for M'.
Using $H(R, M)$: no problem.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:
$S = r^{-1}(H(R, M) - a) \bmod q$.

Signer in this system:
$S = r + H(R, M)a \bmod q$.

Speed advantage:
Skip all inversions.

Security analysis is similar, slightly simpler. See, e.g., 2000 Pointcheval–Stern.

Signature compression

Schnorr signature is
$(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier
recovers $R = B^S/A^h$,
checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs
when $H(R, M)$ is shorter than R.

No security impact:
anyone can compress sigs.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash
functions allow collisions!"

Half-size H output

Schnorr chooses half-size H: e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem: Recall that Schnorr's system is collision-resilient.

Half-size H output

Schnorr chooses half-size H: e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem: Recall that Schnorr's system is collision-resilient.

More serious objection: multi-target preimage attacks.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

Extra H input: $H(R, A, M)$.
Speed impact: negligible.
Alleviates concerns that several public keys could be attacked simultaneously.

Why no signature compression:

1. ECC signatures are short even without compression. 64 bytes for signature using high-security curve.

2. Security of shorter H needs thorough analysis.

3. Double-size H alleviates concerns regarding H security.

4. Avoiding compression allows another speedup: batch signature verification.