ECDSA_CFRG: Schnorr–Kravitz–Vanstone

signatures with a wide-pipe and suffix for high hash-flaw resilience

submitter: Daniel R. L. Brown
presenter: Gaëlle Martin-Cocher

Certicom Research / BlackBerry Standards

CFRG, IETF 93, 2015 July 23

Table of contents

An approach for a new elliptic curve signature scheme

1. Improve on security of both ECDSA and EdDSA.
2. Mathematical security of ECDSA unbroken: do not fix math.
3. Add collision-resilience and other hash-flaw resiliencies.
4. Implementation security fixes:
   - Deterministic signature generation [per CFRG],
   - Side-channel resistance (e.g. constant-time) secret processing.
5. Maintain init-update-finalize processing [per CFRG].
6. Backwards compatible ECDSA verification (not signing) for a given curve.
7. Format agnostic, curve choice (e.g. NIST/Edwards) sets:
   - Coordinate system (and representative)
   - Conversion from point to integer (via byte string)
   - Little-endian or big-endian integer encoding

The resulting proposal: ECDSA_CFRG

Definition (ECDSA_CFRG)

A pair (R, s) is a valid ECDSA_CFRG signature for message M under public key Q if:

$$sR = h_{\text{wide}}(M \| R)G + f(R)Q \tag{1}$$

New over ECDSA:

1. Inclusion of ephemeral in hash input (as suffix).
2. Wide-pipe hash.
3. Full R in signature.
4. Support of Edwards curve formats (not just Weierstrass)
5. Secured signature generation (deterministic, constant-time)

Common notation for Schnorr, ECDSA and ECDSA_CFRG

For base point G, public key Q, message M, a pair (R, s) is a valid Schnorr, ECDSA, ECDSA_CFRG (respectively) signature if

$$sG = R + h(R\|M)Q \tag{2}$$

$$sR = h(M)G + f(R)Q \tag{3}$$

$$sR = h_{\text{wide}}(M\|R)G + f(R)Q \tag{4}$$

(respectively).

- The mathematical design of ECDSA is due to Kravitz and Vanstone.
- Call R ephemeral public key.
- Function f converts points to scalar multipliers.

Talk terminology: aegis, frisk, resilience

Definition

Let

$$Aegis = Age \times Eyes \tag{5}$$

Definition

Formal risk is

$$\sum_{\text{threat}} \text{Probability[threat viable]} \times \text{Damage(threat launched)}. \tag{6}$$

To frisk is to formalize risk.

Definition (Resilience)

Given some threat t, let t-resilience be a reduced risk of an attack from threat t, by either low probability or low damage.

How Schnorr depends on hash security

Theorem (Pointcheval–Stern)

Schnorr signatures secure if discrete logs secure and hash is random oracle.

- Random oracle is optimistic security for hash.
    - Theorem is not evidence of resilience: using suffixed point and narrow-pipe hash not really collision-resilient.
- Loose reduction: strictly applies only for doubled group size.
- Trust (verified?) that proofs work if hash also used for deterministic ephemeral key generation.
- Hash—throughout this presentation—is effectively conventional bit-string-output hash (e.g. SHA-384) after modular reduction.
    - All security properties defined over reduced hash function.

Hash deletion

Definition (Hash deletion)

Message M' and hash value h' such that

$$h(R \| M') = h' \tag{7}$$

for a non-negligible fraction of all R of a given length.

Theorem (2015)

Given a hash deletion, for any s' and public key Q,

$$(R', s') = (s'G - h'Q, s') \tag{8}$$

is a valid Schnorr signature on M': a signer-absent forgery.

Implicit aegis of deletion attacks

Theorem (2015)

Collision-resistant (and random oracle) hashes are deletion-resistant.

Definition (Nested hash deletion)

Message M' and hash value h' such that $h(S\|h(R\|M')) = h'$ for all R, S (with non-negligible chance).

Theorem (2015)

If HMAC secure, then hash is nested-deletion-resistant.

Theorem (2015, Circular self-aegis)

If Schnorr signatures secure, then hash is deletion-resistant.

Deletion's aegis deficiencies?

1. Lack of incentives:
    1. Not among holy grail of collision, preimage, second preimage.
    2. Deletion-resistance is an off-label claim (hash as cure-all not path to resilience: see earlier comment).
    3. No specific mention of deletion in SHA-3 competition.
    4. Deletion attacks strictly only relevant to Schnorr.
2. Deletion attacks not known to imply:
    1. Preimage attacks
    2. Second preimage attacks
3. Lack of partial attacks (⇒ lack of evidence of effort):
    1. No MD5 or SHA0 deletion attacks.
    2. No published reduced-round hash deletion attacks.

Deletion threat viability?

1. Joux multicollisions? Kelsey–Kohno herding?
   - Deletion implies a multicollision: a very wide one!
   - Wide multicollisions not much harder than collisions, against iterated hashes like SHA-2.
   - Deletion not directly targeted: so these attacks provide no aegis for deletion.
2. Deletion attacks on some iterated hash corresponds to weak key some compression functions:
   - Message block W' such that $E_{W'}(H) = C \boxminus H$ for some constant C.
   - Targeted deletion: slight aegis for deletion (my eye).
3. Heed these as early warning signs?
4. What is the security level of deletion-resistance?
   - Pessimist: same as (multi)collision-resistance.
   - Optimist: infinity, maybe no (h', M') exists.

How Schnorr depends on hash security, revisited

Theorem (Neven, Smart and Warinschi)

Schnorr signatures secure in generic group model if hash has chosen-target random-prefix (second) preimage resistance (RP(S)P).

- RP(S)P attacks has implicit aegis similar to deletion attacks:
  - Schnorr signatures need RP(S)P secure hash to avoid forgery
  - Collision resistant hashes are RP(S)P secure.
- RP(S)P related to ePre (keyed) hash security of Rogaway, Stam and others: so it has a little extra aegis.

Now, how deletion-resilient is ECDSA?

Theorem (2015)

Given a deletion attack and signer who will sign attacker chosen message, adversary can forge a related message.

Proof.

Deletion leads to collision: $h(R_1||M') = h(R_2||M') = h'(M')$. Exploit ECDSA's lack of collision-resilience. Ask signer to sign $R_1||M'$. Same signature also valid for unsigned $R_2||M'$. □

Mitigating damages from ECDSA deletion

Signer-present mitigations to deletion for ECDSA but not for Schnorr:

1. Damage of forgery limited to R_2: else, if signer willing to sign $R_1||M'$, why not just get the signer to sign $R_2||M'$?
2. Inspect $R_1||M'$ for suspicious content: thwarts weakest deletion attacks with odd looking messages,
3. Opt to control content of signed messages, such as prefix, hindering deletion attack: some signers already do this to ward off potential collision-extension attacks.
4. Track messages signed to repudiate those resulting from deletion/collision attack.

Hash Zeroizers

Definition

A message M' such that $h(M') = 0$.

Theorem (2001)

A hash zeroizer leads to an odd-message signer-absent ECDSA forger. (The zeroizer message is forged.)

Theorem (2001)

A collision-resistant hash has at most one zeroizer message.

Domain parameter attacks

Theorem (Vaudenay)

If the elliptic curve order is chosen maliciously, then the hash can have a zeroizer or a collision, and consequently, ECDSA is forgeable.

Countermeasures:

1. Well-chosen elliptic curves.
   - Only known ways to find prime-field curves of order:
     - Exhaustive search.
     - Complex multiplication.
2. Truncate hash to smaller than order n of G.

One-up problem

Definition (One-up problem (2008))

Given two points A and B, find a point C such that:

$$C = A + f(C)B \tag{9}$$

Theorem (2008)

Given a one-up problem solver S, a signer-absent all-message ECDSA forger F^S can be constructed.

- Pessimist: 1-up requires $m \ll \sqrt{n}$ group ops (low aegis).
- Optimist: 1-up requires $n \gg \sqrt{n}$ group ops (best known attack).
- Theorist: 1-up secure in generic group model (implicit in next).

Kravitz–Vanstone signature mathematical security

Theorem (2002)

Kravitz–Vanstone signature (e.g. ECDSA) with the zeroizer-resistant hash in the generic group model resists forgery of the type specified below according additional security of the hash:

Forger type		Hash
Signer	*Messages forgeable*	*Additional security*
Absent	*All*	*None*
Absent	*Odd*	*Preimage*
Present	*All*	*Second preimage*
Present	*Odd*	*Collision*

Kravitz–Vanstone signature mathematical security

Theorem (2005)

Kravitz–Vanstone (e.g. ECDSA) signatures with a random oracle hash resist all types of forgers if they resist signer-absent all-message forgers (but with a cost factor proportional to the number of hash queries).

Risk comparison: Schnorr versus Kravitz–Vanstone

Formal risk terms due to deletion:

- Schnorr:

$$\ldots + \Pr[\text{hash deletion}] \times \text{Damage}(\text{signer-absent forgery}) + \ldots \quad (10)$$

- Kravitz–Vanstone (i.e. DSA/ECDSA):

$$\ldots + \Pr[\text{hash collision}] \times \text{Damage}(\text{signer-present forgery}) + \ldots \quad (11)$$

Comparison:

$$\underbrace{\textit{smaller} \times \textit{larger}}_{\text{Schnorr}} \underbrace{\approx}_{?} \underbrace{\textit{small} \times \textit{large}}_{\text{ECDSA}} \quad (12)$$

Other terms in formal risk from other threats: also hard to compare.

Idea: Schnorr–Kravitz–Vanstone

- Use R thrice in the verification:

$$s \underbrace{R}_{\text{ElGamal}} = \underbrace{h(R\|M)}_{\text{Schnorr}} G + \underbrace{f(R)}_{\text{Kravitz–Vanstone}} Q \tag{13}$$

  where, $r = f(R)$, is a multiplier (scalar, integer) mapped from a point R via a byte string (and a field coordinate).
- Novel combination of apparent resiliencies:
  - Collision
  - Deletion
  - Zeroizer
  - RP(S)P
  - One-up
- Lacks simultaneous message-dependent ephemerals and IUF processing.

The proposal: ECDSA_CFRG

Definition (ECDSA_CFRG)

A pair (R, s) is valid ECDSA_CFRG signature for message M under public key Q if:

$$sR = h_{\text{wide}}(M \| R)G + f(R)Q \tag{14}$$

Improvements over basic Schnorr–Kravitz–Vanstone idea:

1. Places R in suffix instead of prefix (IUF processing).
2. Uses wide-pipe hash (collision-resilience with suffix).
3. Signature has R not $r = f(R)$ (batching and faster verify), unlike old ECDSA standards.

Message digesting?

1. For 256-bit or less curves (Curve25519, P256) use SHA-384:
    1. Widely available hash.
    2. Wide pipe (512 bits).
    3. Truncate to bit length of n, per existing ECDSA. (Almost defends against domain parameter attacks: but ECDSA_CFRG has other defenses.).
2. For 512-bit or less curves, use SHA3-512.
    - More trusted hash: better aegis (newer than SHA2, but more eyes, and arguably more trusted origin).
    - Wide pipe: 1024 bits.
    - Truncate output to one bit less than n, to better avoid domain parameter attacks.
3. For 521-bit curves, use SHA3-512:
    - Pipe almost wide enough (1024 bits nearly 1042 bits).
    - Same reasons as above.
4. For other curves, seek wider-pipe hash.

ECDSA_CFRG compared to ECDSA and Schnorr

Theorem (Pointcheval–Stern?)

If discrete logs secure and hash is random oracle, then ECDSA_CFRG is unforgeable.

Theorem (2015)

If ECDSA is signer-absent unforgeable, then ECDSA_CFRG is signer-absent unforgeable.

Theorem (2015? — RETRACTED: NO LONGER CLAIMED)

RETRACTED: If contrived-hash Schnorr signatures unforgeable, then ECDSA_CFRG is signer-absent, all-message, deterministic unforgeable. (RETRACTED)

Koblitz–Menezes (2015): informally, generic group model security proofs for ECDSA also apply to ECDSA_CFRG.

ECDSA_CFRG signature generation

1. Bias and correlated ephemeral signing keys leak static key: so try to apply all protections with high aegis.
2. Multiplicative masking for the multiplier inversion: do not invert k directly.
3. Use constant-time scalar multiplication kG, additive masking.
4. Key derivation:
   - Keyed pseudorandom function (PRF)
   - Variable-input length, constant output-length
   - Not just a simple hash (extensions mean not strict PRF).
5. Static signing key derived from seed: $d = \mathrm{PRF}_{seed}(0^8)$.
6. Ephemeral signing key co-derived from seed and message.

$$k = \mathrm{PRF}_{seed}(1^8 \| M). \tag{15}$$

Key separation through PRF properties, not ad hoc ROM optimism.

Key derivation function

- Try to output a fixed extra amount bits, at least 50%.
- For curves with n of 256 bits or less:

$$\mathrm{PRF}_{seed}(data) = \text{HMAC-SHA2-384}(seed, data) \tag{16}$$

  with no HMAC truncation.
  - HMAC widely believed to be a good PRF (aegis).
  - Harmony with message digest SHA2-384.
- For n with $m \in [257, 571]$ bits, let $L = \lceil 3m/2 \rceil$ and:

$$\mathrm{PRF}_{seed}(data) = \text{SHAKE-256}(L, seed \| \operatorname{encode}(L) \| data) \tag{17}$$

  - SHAKE is next generation of key derivation.
  - First SHAKE input is output length.
  - Length included in SHAKE input to help avoid truncation.
  - Harmony with message digest SHA3-512.

Backwards compatibility with ECDSA

1. Backwards compatible verification:
   - Given same curve (and curve formatting),
   - ECDSA_CFRG signature (R, s) of message M can converted to ECDSA signature $(r, s) = (f(R), s)$ of message $M||R$.
   - Verifiable with existing ECDSA verifiers, which is good: for codesigning upgrades, etc.
2. Backwards incompatible signing:
   - Incompatible signing is good, because any existing insecure ECDSA signer cannot be re-used for ECDSA_CFRG.
   - Active steps would have be taken to modify an ECDSA signer to build an ECDSA_CFRG signer:
     1. Dig past the (r, s) interface to find internal value of R.
     2. Find R before s is computed.
     3. Append R suffix to message before computing s.
     4. Update to a wide-pipe hash.
3. Forwards incompatible verification: good and obvious.

ECDSA_CFRG and ECDSA internal verification

```
// ECDSA_CFRG pseudocode spec only: not for real use
# include "ecdsa_cfrg.hh"
# define VERIFY  verify // verify_via_ecdsa
static bool verify (point Q, bits M, ecdsa sig)
{
  return sig.r == f((hash(M)*G + sig.r*Q) / sig.s) ;
}
static bool verify (point Q, bits M, ecdsa_cfrg sig)
{
  return sig.s*sig.R == hash(M||sig.R)*G + f(sig.R)*Q ;
}
static bool verify_via_ecdsa (point Q, bits M, ecdsa_cfrg sig)
{
  return verify (Q, M||sig.R, (to_ecdsa)(sig)) ;
}
```

External interface

```
bool verify_ecdsa_cfrg (bits Q, bits M, bits sig)
{
  return VERIFY((to_point)(Q), M, (to_ecdsa_cfrg)(sig)) ;
}
bits verification_key_ecdsa_cfrg (bits key)
{
  return (to_bits)(prf(key) * G) ;
}
bits sign_ecdsa_cfrg (bits key, bits M)
{
  mult  k = prf(key, M) ;
  mult  m = prf(key, k) ; // mask leaky / op
  mult  d = prf(key) ;
  point R = k*G ;
  mult  s = (hash(M||R) + f(R)*d) / (k*m) ;
  return (to_bits)((ecdsa_cfrg){R , (s*m)}) ;
}
```

File “ecdsa_cfrg.hh” (types and conversions)

```
typedef struct { /*...*/ } mult ;
typedef struct { /*...*/ } point ;
typedef struct { /*...*/ } bits ;
typedef struct {point R; mult s;} ecdsa_cfrg ;
typedef struct {mult  r; mult s;} ecdsa ;
point G = { /*...*/ } ;
bits       to_bits  (point) ;
point      to_point (bits) ;
bits       to_bits       (ecdsa_cfrg) ;
ecdsa_cfrg to_ecdsa_cfrg (bits) ;
mult       to_mult       (point P)
/* Weierstrass: { return (to_mult)((to_bits)(x(P)));} */ ;
# define f to_mult
ecdsa      to_ecdsa      (ecdsa_cfrg sig)
{
  return (ecdsa){f(sig.R), sig.s} ;
}
```

File “ecdsa_cfrg.hh” (primitive operations)

```
mult hash (bits) ;
mult prf (bits) ;        // MUST:   deterministic
mult prf (bits, bits) ; // SHOULD: deterministic
mult prf (bits, mult) ; // SHOULD: non-deterministic
bool operator == (mult, mult);
mult operator  + (mult, mult);
mult operator  * (mult, mult);
mult operator  / (mult, mult);
bits operator || (bits, bits);
bits operator || (bits M, point R)
{    return M || (to_bits)(R) ; }
bool operator == (point, point);
point operator + (point, point);
point operator * (mult , point);
point operator / (point, mult ); // ECDSA verify spec only
```