Signatures: Ilari's proposal

- A modified version of EdDSA

Introduction: EdDSA

- An advanced digital signature primitive, encompassing lots of improvents to original ElGamal Scheme
- No inversions modulo order
- Deterministic

Introduction: Why not original EdDSA

- Orig. EdDSA does not support IUF
- Curve requirements not met by Orig. EdDSA
- Requires exotic hashes at >255 bits
  - >512 bit output and such
- Later revision of EdDSA addressed first two
  - And proposed modifications to address third

Adding IUF

- IUF via prehashing
  - Some protocols assume hash signing
- Add firewalling (sign the hash used)
  - Prevents cross-hash attacks
  - Widely used (but broken) RSA PKCS#1 v1.5
  - Put into main signature to avoid bypass
  - There can be value for identity (offline signature)
  - Obviously not helpful if internal hash really broken

More curves

- One can add support for more curves by modifying point encoding.
  - Little-endian Y enough-bits + X-sign bit encoding.
  - Can encode any prime Edwards curve
- Allows all sorts of curves (not all good)

Using ordinary hashes (#1)

- EdDSA specifies double-width hash
  - Seemingly great overkill.
  - 64 extra bits should be enough.
    - Enough to reach 448-bits with 512-bit hash.
    - Enough for 2^128 signatures with arbitrary curve
    - Enough for sqrt(l) signatures with near-POT curve
- Nevertheless use more for arbitrary curves
  - 1/4 of bits, to reach sqrt(l) signatures.
- Schnorr croaks at sqrt(l) signatures.

Using ordinary hashes (#2)

- Seed/a generation in EdDSA is via splitting
  - So generate seed and a separatedly.
  - Some sort of labeled PRF.
  - Seed/a might get stored in private key.

Limits of ordinary hashes

- Ordinary hashes cap at 512 bits
  - Limits curves to 448 bits for near-PoT
  - Limits curves to 409 bits for arbitrary.
- Few curves outside these limits
  - Ridinghood (in fact, seems OK)
  - E-521 (definitely not OK)
- PRF would be possible but extra complexity
  - Really avoid nonstandard APIs.

Extras: Personalization

- Cross-protocol attacks are a perenial problem
- Difficult to avoid in protocol design
- One way to deal with it: Personalization
- Signer and verifier need to agree context
  - Specified by protocol, arbitrary length
    - Idea: <Protocol> <version>, <role in protocol>
      - E.g. ”TLS 1.2, Client Certificate Verify”
- Use the same encoding as hash/message
  - Unaligned hashes, but those should be OK

Design: Some other approaches

- Double-pipe seed|m, m|rB.
  - Requires signer to pick key beforehand.
  - Doesn't fit protocols assuming hash-signing.
- Forking: m|pad|seed, m|pad|rB.
  - Is this even secure?
  - Pad has to be picked carefully.
  - Needs somewhat non-standard hash interface.
  - Again, same fit problems as above.