# 4 YEARS
# of violent programming

# Language

## C++ 03

+ no exceptions
+ no STL
+ no RAII
+ no RTTI

# Language

## C++ 03

+ no exceptions
+ no STL
+ no RAII
+ no RTTI

## C 99

+ compiles up to 2x faster
+ has stable ABI
+ is portable
+ has `restrict`

# Architecture .SOLID

*Ignoring SOLID* leads to:

+ bulky methods
  with confusing lists
  of parameters
+ high degree
  of code duplication
+ monolithic architecture

# Architecture .SOLID

```
// CU 2.x
// Six different implementations of CFB mode!
// You've got to be kidding.

status // gost/src/cipher_gost_89_cfb.cpp
CuCipherGost89CfbDecodeUpdate(
  ctx, sbox, key, in[inlen], out[], *len);

status // gost_simd/src/cipher_gost_89_sse.cpp & ./cipher_gost_89_neon.cpp
CuCipherGost89FastCfbDecodeUpdate(
  ctx, sbox, key, in[inlen], isFinal, out[], *len);

status // gost_simd/src/cipher_gost_89_avx.cpp
CuCipherGost89Fast256CfbDecodeUpdate(
  ctx, sbox, key, in[inlen], isFinal, out[], *len);

status // fips/src/cipher_aes256_cfb128.cpp
CuCipherAES256CFB128DecodeUpdate(
  ctx, key, in[len], out[]);

status // fips_simd/src/cipher_aes256_ni_cfb128.cpp
CuCipherAesNi256Cfb128DecodeUpdate(
  ctx, key, in[len], out[]);
```

# Architecture .SOLID

*~~Ignoring SOLID~~* leads to:

+ bulky methods with confusing lists of parameters
+ high degree of code duplication
+ monolithic architecture

*Applying SOLID* allows for:

+ simple methods
+ no code duplication
+ modular architecture

# Architecture .SOLID

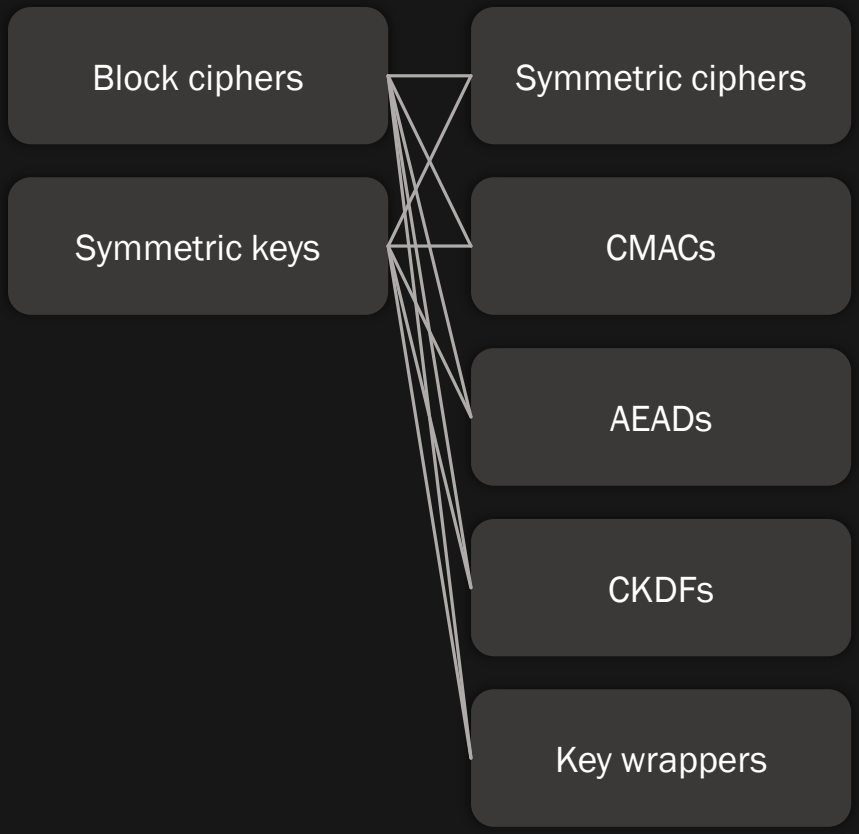
```
// CU 3.x
// Works for any mode {ECB, CBC, OFB, CFB, CTR, CNT}
// Works for any cipher {g89x-impl-bc, magma-impl-bc, kuznyechik-impl-bc}
// Works for any implementation {universal, sse2, ssse3, avx2, neon}
// Works for any section size {meshed, unmeshed}

// Initialises any symmetric cipher
status
initialiseSc(ctx, key, Iv[]);

// Updates any symmetric cipher
status
updateSc(ctx, out[], in[]);
```

# Architecture .SOLID

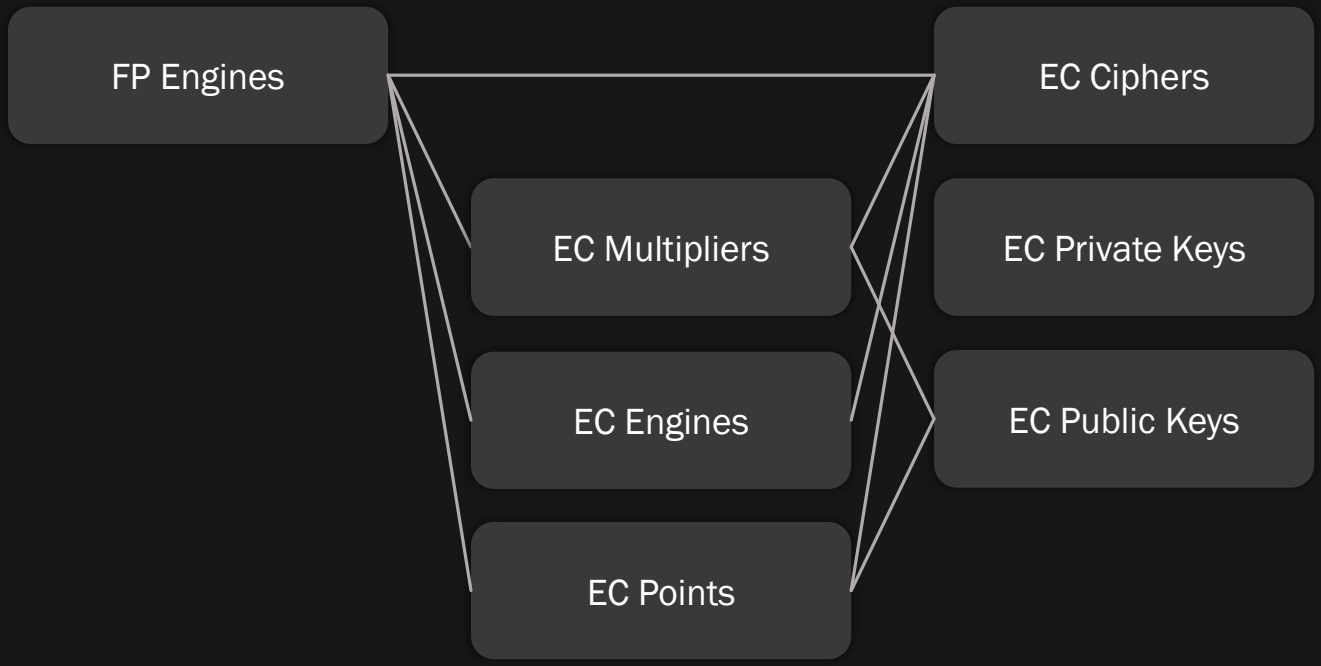
*Same goes for:*

+ allocators,
+ generators,
+ symmetric and asymmetric keys, key wrappers,
+ block ciphers, CMACs, and CKDFs,
+ symmetric ciphers, and AEAD ciphers,
+ hashes, HMACs, and HKDFs,
+ elliptic curves.

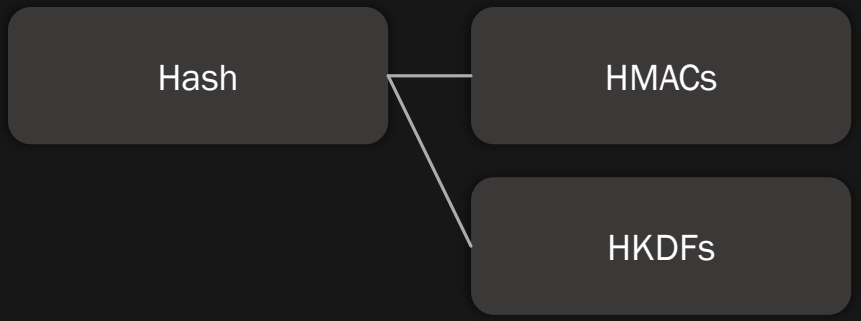
# Architecture .Hierarchies



# Architecture .Hierarchies



# Architecture .Hierarchies



# Architecture .KISS

*Post-knowledge and KISS enabled:*

+ simple and uniform APIs,
+ very little to none quirks in most use cases,
+ less built-in monkey patches,
+ code which is harder to shoot yourself in the foot with.

# Architecture .KISS

```
// CU 2.x
// Initializes prng-advanced with entropy
status
CuPrngAdvancedInitialize(ctx, initializer[]);

// CU 2.x
// Initializes prng-iv with what?
status
CuPrngIvInitialize(ctx, initSip[], initRand[]);


// CU 3.x
// Works for any generator {stdlib-gn, xs128-gn, rfci366-x-gn, tc26-x-gn}

// Initialises any generator with entropy
status
initialiseGn(ctx, entropy[]);

// Initialises any generator with another generator
status
bootstrapGn(ctx, starter);
```

# Architecture .KISS

```
// CU 2.x
// This method produces masked symmetric key
// while there is no available public API to unmask it
status
CuKeySymmetricDerivationGost12S256To256(
  ctx, rngCtx, paramset, Ukm[], privateKey, publicKey, keys[], masks[]);


// CU 3.x
// This method produces plain secret,
// which can be loaded into a key or a cipher
status
exchangeKeysWithEcCipher(
  ctx, privateKey, publicKey, hash, secret[], nonce[]);
```

# Architecture .Objects

*Life cycle* of object:

1. Construct (create) object
2. Work with object
   - `initialise`, update, `finalise`, etc.
   - `set`, `get`, `generate`, etc.
   - `serialise`, `deserialise`
   - `clone`
   - `sanitise`
3. Destroy object

# Architecture .Objects

```
// CU 3.x
// Evaluating OMAC1(message, secret) with SSE2 version of Kuznyechik

Cmac = &ItcsCu_Omac1Cmac;
cipher = &ItcsCu_KuznyechikSse2Bc;
key = getBcKeyTraits(cipher);

// Constructs key and MAC
CHECK(createSk(allocator, &key));
CHECK(createCmac(allocator, &Cmac, cipher));

// Sets key and computes MAC
CHECK(setSk(key, secret[], randomiser));
CHECK(initialiseCmac(Cmac, key));
CHECK(updateCmac(Cmac, message[]));
CHECK(finaliseCmac(Cmac, digest[]));

// Destroys key and MAC
CHECK(destroyCmac(allocator, Cmac));
CHECK(destroySk(allocator, key));
```

# Architecture .Objects

```
// CU 3.x
// Signing message with random key on TC26-B 256-bit curve

curve = &ItcsCu_Gost12x256Tc26BEcCipher;
key = getEcCipherPrivateKeyTraits(curve);

// Constructs key and curve
CHECK(createEcPrk(allocator, &key));
CHECK(createEcCipher(allocator, &curve, 0));

// Initialises curve, generates key, and signs the message
CHECK(initialiseEcCipher(curve));
CHECK(generateEcPrk(key, curve, generator, NULL));
CHECK(signWithEcCipher(curve, key, noncer, signature[], message[]));

// Destroys key and curve
CHECK(destroyEcCipher(allocator, curve));
CHECK(destroyEcPrk(allocator, key));
```

# Architecture .Objects

Runtime
*(public, non-static)*

Metadata
*(public, static)*

VMTs
*(public, static)*

Context
*(private, non-static)*

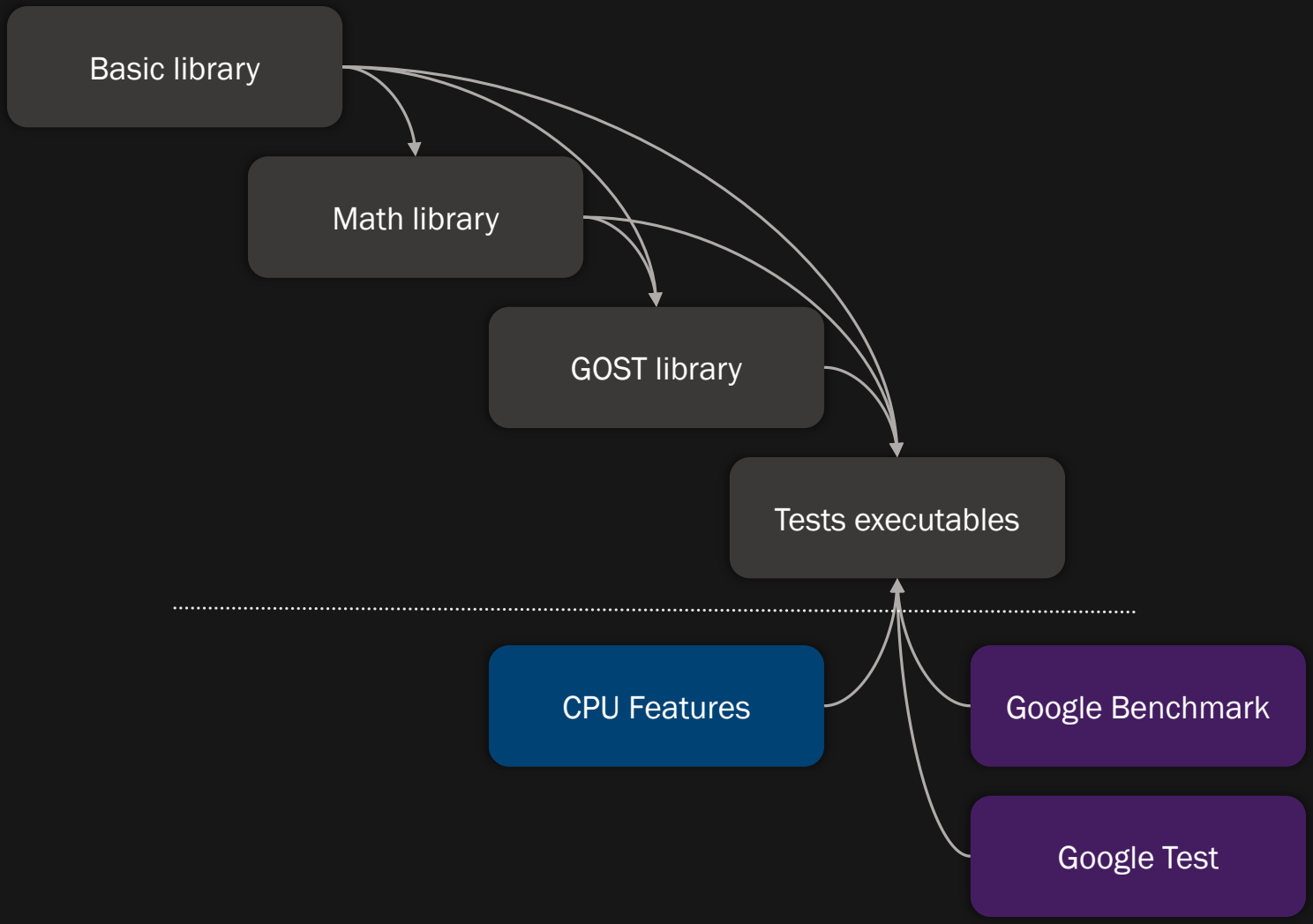
# Architecture .Contracts

| | Debug | Release |
|---|---|---|
| Programming errors (via unsafe API) | Contracts *(debug breaks)* | None |
| Programming errors (via safe API*) | Return codes | Return codes |
| Runtime errors | Return codes | Return codes |

# Architecture .Contracts

```
// CU 3.x
// Real-life example of contracts.
// Execution will break in case of breach in debug flavour.
// No-ops in release flavour.

ItcsCu_status_t
intl_MgmAcInitialise(
     ItcsCu_authenticatedCipher_t *self,
     ItcsCu_symmetricKey_t const *key,
     uint8_t const nonce[],
     size_t size,
     ItcsCu_generator_t *randomiser /* optional */
) {
     ASSUME_TRUE(isInstantiatedAc(self));
     ASSUME_TRUE_UNLESS(isInitialisedBc(getAcBc(self)), key);
     ASSUME_EQUAL(size, getBcBlockSize(getAcBc(self)));
     ASSUME_VALID(nonce, size);

     ...
```

# Architecture .Dependencies



# What's new .Mechanisms

*New generation of core introduces features:*

+ magma-{universal, ssse3, neon, avx2}-bc
+ kuznyechik-{universal, sse2, sse2-m, avx2}-bc
+ ctr-{encryption, decryption}-sc
+ omac1-cmac
+ mgm-{universal, clmul}-{encryption, decryption}-ac
+ kexp15-kw
+ sp800-ckdf
+ sp800-hkdf

# What's new .Performance

*New generation of core provides somewhat better performance*

+ `g89x-universal-bc` is up to 20% faster,
+ `rfci366-x-gns` and `tc26–x-gns` are 1.5x to 3x faster,
+ `g94x-ssse3-hs` is up to 3x faster,
+ signatures are up to 3x faster[1].

[1] with default precomputation level

# What's new .Performance

*Options:*

- `--enable-sse`
- `--enable-avx2`
- `--enable-neon`
- `--enable-aesni`
- `--enable-simd` (?)

# What's new .Performance

*Options:*

- `--enable-sse`
- `--enable-avx2`
- `--enable-neon`
- `--enable-aesni`
- `--enable-simd (?)`

*Refined options:*

- `--enable-hwaccel`
  *(for hardware acceleration)*
- `--enable-autovec`
  *(for automatic vectorization)*

# Testing .Conduct

*Code of conduct includes:*

+ no warnings (compiling with `/W4`, `-Wall -Wextra`),
+ no leaks, no uninitialised accesses (`valgrind`),
+ no unaligned accesses (CI on ARM),
+ less than 0.5s of smoke tests with basic coverage,
+ more than 5000 unit tests with 97% coverage.

# Testing .Benchmarks

*Benchmarking binary* provides:

+ single-point entry for all benchmarks,
+ complete and honest set of tests,
+ uniform and predictable naming scheme,
+ multiple ranges of threads.

# Future .Architecture

Uniform building
*(CMake, Meson, etc.)*

Uniform delivery
*(Conan, Ivy, etc.)*

Uniform conformance testing
*(Datafit, NIST CAVP, etc.)*

Built-in CPU features detection

# Future .Implementation

Safe APIs

Better Windows builds
*(~~MSVC~~ Clang and/or ~~MSBuild~~ Ninja)*

Bindings to C++

# Future .CI&CD

Automatic redeployment

`##teamcity buildStop` *in TeamCity 2019*

`${ivy.deps.changes}` *in Ivy*

Build → Test → Deploy CI chains

;)