

C++ static code analysis:

Cryptographic keys should be robust

5-6 minutes

Most of cryptographic systems require a sufficient key size to be robust against brute-force attacks.

[NIST recommendations](#) will be checked for these use-cases:

Digital Signature Generation and Verification:

- $p \geq 2048$ AND $q \geq 224$ for DSA (p is key length and q the modulus length)
- $n \geq 2048$ for RSA (n is the key length)

Key Agreement:

- $p \geq 2048$ AND $q \geq 224$ for DH and MQV
- $n \geq 224$ for ECDH and ECMQV (Examples: secp192r1 is a non-compliant curve ($n < 224$) but secp224k1 is compliant ($n \geq 224$))

Symmetric keys:

- key length ≥ 128 bits

This rule will not raise issues for ciphers that are considered

weak (no matter the key size) like DES, Blowfish.

Noncompliant Code Example

[botan](#)

```
#include <botan/dl_group.h>
#include <botan/ec_group.h>
#include <botan/pubkey.h>
#include <botan/rng.h>
#include <botan/rsa.h>

// RSA
std::unique_ptr<Botan::RandomNumberGenerator> rng(new
Botan::System_RNG);
Botan::RSA_PrivateKey rsaKey(*rng, 1024); //
Noncompliant; 2nd argument "bits" should be  $\geq 2048$ 

// DSA / DH
Botan::DL_Group("modp/ietf/1024"); // Noncompliant; 1st
argument "name" last component should be  $\geq 2048$ 
Botan::DL_Group("dsa/botan/1024"); // Noncompliant; 1st
argument "name" last component should be  $\geq 2048$ 

// EC
Botan::EC_Group("secp160k1"); // Noncompliant; EC
key length is 160. Should be  $\geq 224$ 
```

[crypto++](#)

```
#include <cryptopp/dh.h>
#include <cryptopp/oids.h>
```

```
#include <cryptopp/rsa.h>
#include <cryptopp/rng.h>
#include <cryptopp/osrng.h>
```

```
CryptoPP::AutoSeededRandomPool rng;
```

```
// RSA
```

```
CryptoPP::InvertibleRSAFunction params;
params.GenerateRandomWithKeySize(rng, 1024); //
Noncompliant; 2nd argument "keySize" should be  $\geq 2048$ 
```

```
// DSA
```

```
CryptoPP::DSA::PrivateKey privateKey;
privateKey.GenerateRandomWithKeySize(rng, 1024); //
Noncompliant; 2nd argument "keySize" should be  $\geq 2048$ 
```

```
// DH
```

```
CryptoPP::DH dh;
dh.AccessGroupParameters().GenerateRandomWithKeySize(rnd,
1024); // Noncompliant; 2nd argument "keySize" should be
 $\geq 2048$ 
```

```
// EC
```

```
CryptoPP::ASN1::secp112r1(); // Noncompliant; EC key
length is 112. Should be  $\geq 224$ 
```

[OpenSSL](#)

```
#include <openssl/dh.h>
#include <openssl/dsa.h>
```

```
#include <openssl/ec.h>
#include <openssl/obj_mac.h>
#include <openssl/rsa.h>

// RSA
RSA_generate_key_ex(key, 1024, e, NULL); //
Noncompliant; 2nd argument "bits" must be  $\geq 2048$ 

// DSA
DSA_generate_parameters_ex(dsa, 1024, NULL, 0, NULL,
NULL, NULL); // Noncompliant; 2nd argument "bits" must be
 $\geq 2048$ 

// DH
DH_generate_parameters_ex(dh, 1024,
DH_GENERATOR_2, NULL); // Noncompliant; 2nd
argument "prime_len" must be  $\geq 2048$ 

// EC
EC_KEY_new_by_curve_name(NID_secp112r1); //
Noncompliant; EC key length is 112. Should be  $\geq 224$ 
```

Compliant Solution

[botan](#)

```
#include <botan/dl_group.h>
#include <botan/ec_group.h>
#include <botan/pubkey.h>
#include <botan/rng.h>
```

```
#include <botan/rsa.h>
```

```
std::unique_ptr<Botan::RandomNumberGenerator> rng(new  
Botan::System_RNG);
```

```
// RSA
```

```
Botan::RSA_PrivateKey rsaKey(*rng, 2048); // Compliant;  
2nd argument "bits" is  $\geq 2048$ 
```

```
// DSA / DH
```

```
Botan::DL_Group("modp/ietf/2048"); // Compliant; 1st  
argument "name" last component is  $\geq 2048$ 
```

```
Botan::DL_Group("dsa/botan/2048"); // Compliant; 1st  
argument "name" last component is  $\geq 2048$ 
```

```
// EC
```

```
Botan::EC_Group("secp224k1"); // Compliant; EC key  
length is 224.
```

[crypto++](#)

```
#include <cryptopp/dh.h>
```

```
#include <cryptopp/oids.h>
```

```
#include <cryptopp/rsa.h>
```

```
#include <cryptopp/rng.h>
```

```
#include <cryptopp/osrng.h>
```

```
CryptoPP::AutoSeededRandomPool rng;
```

```
// RSA
```

```
CryptoPP::InvertibleRSAFunction params;  
params.GenerateRandomWithKeySize(rng,2048); //  
Compliant; 2nd argument "keySize" is  $\geq 2048$ 
```

```
// DSA
```

```
CryptoPP::DSA::PrivateKey privateKey;  
privateKey.GenerateRandomWithKeySize(rng, 2048); //  
Compliant; 2nd argument "keySize" is  $\geq 2048$ 
```

```
// DH
```

```
CryptoPP::DH dh;  
dh.AccessGroupParameters().GenerateRandomWithKeySize(rnd,  
2048); // Compliant; 2nd argument "keySize" is  $\geq 2048$ 
```

```
// EC
```

```
CryptoPP::ASN1::secp256r1(); // Compliant; EC key length  
is 256
```

[OpenSSL](#)

```
#include <openssl/dh.h>  
#include <openssl/dsa.h>  
#include <openssl/ec.h>  
#include <openssl/obj_mac.h>  
#include <openssl/rsa.h>
```

```
// RSA
```

```
RSA_generate_key_ex(key, 2048, e, NULL); // Compliant;  
key size  $\geq 2048$ 
```

// DSA

```
DSA_generate_parameters_ex(dsa, 2048, NULL, 0, NULL,  
NULL, NULL); // Compliant; key size  $\geq$  2048
```

// DH

```
DH_generate_parameters_ex(dh, 2048,  
DH_GENERATOR_2, NULL); // Compliant; "prime_len" is  $\geq$   
2048
```

// EC

```
EC_KEY_new_by_curve_name(NID_secp224r1); //  
Compliant; EC key length is 224
```

See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [Mobile AppSec Verification Standard](#) - Cryptography Requirements
- [OWASP Mobile Top 10 2016 Category M5](#) - Insufficient Cryptography
- [NIST 800-131A](#) - Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
- [MITRE, CWE-326](#) - Inadequate Encryption Strength

