FIRSTBRIDGE

FBCrypto: Elliptic Curves Made Easy

Oleksiy Lukin
alukin@gmail.com

Why I have to switch to ECC?

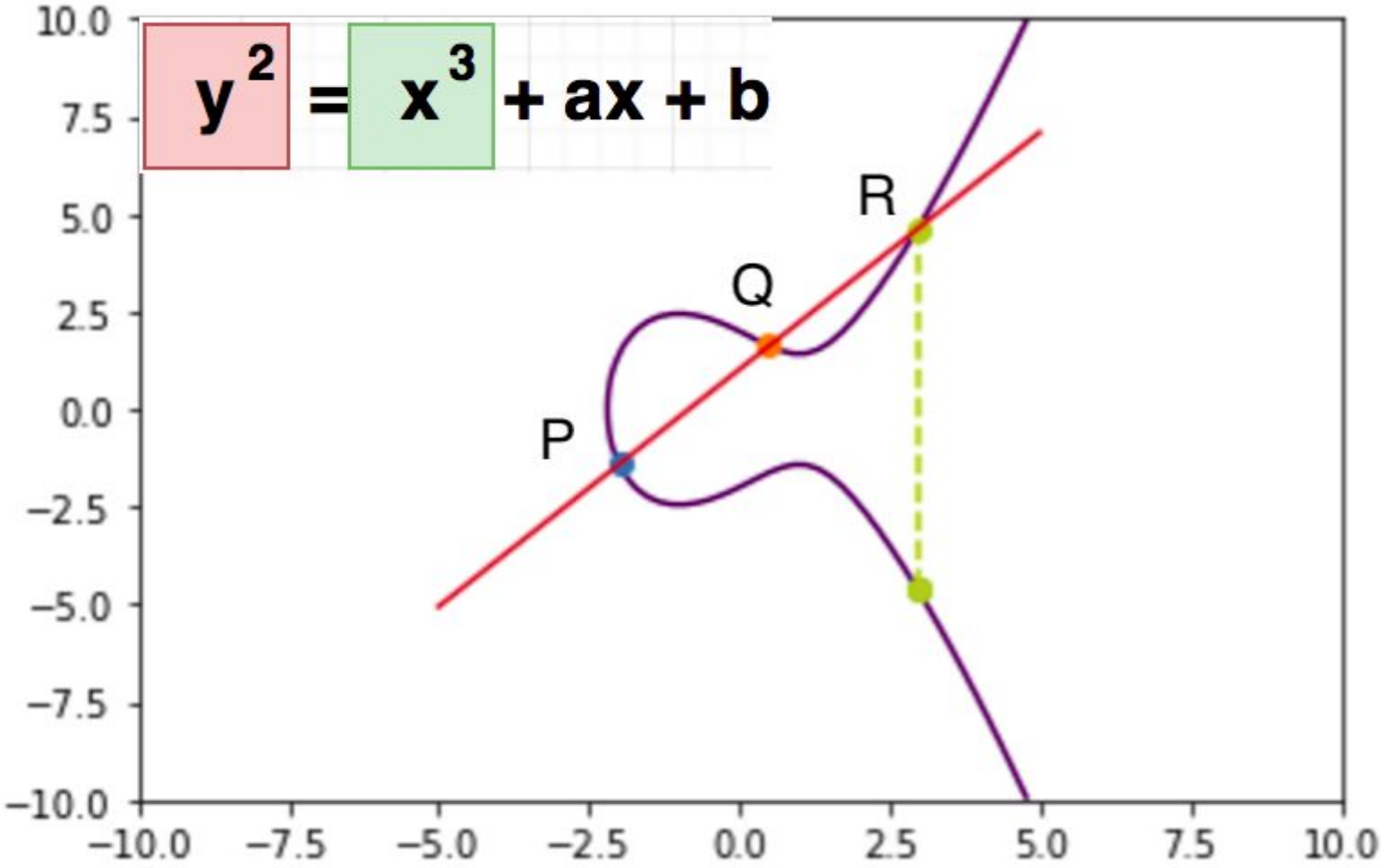
Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

ECC vs RSA: what's different?

- Smaller key lenght, faster implementations
- RSA: Sign, verify, encrypt, decrypt (slow, small)
- ECC: Sign and verify only, different integrated encryption schemes, key agreements.
- RSA: exonetiantion, slow on big keys
- ECC: polinominal operations, much faster
- http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/
- https://hackernoon.com/how-does-rsa-work-f44918df914b

Simple elliptic curve



What is FBCrypto

- Libraries with simple interface to complex symmetric and assymetric cryptography
- Coherent implementations in Java, JavaScript, C++ with common tests
- Coherent parameter sets for base key length (160-521 bit)
- Corect key generators, readers/writers, X.509 certificate parsers, etc.
- Utility for ECC X.509 CSR generation and other common operations

Goals of FBCrypto development

- Encourage strong cryptography usage in OpenSource products
- Simple interface that does not force programmer to in-depth study of cryptograpy
- Military grade modern cryptography behind simple interface
- Switchable implementations (including hardware backend), configurable key lenghth
- “The same thing” in Java, C++, JavaScript with similar interface, compatible formats, common test data and interoperabilty tests

Why opensource?

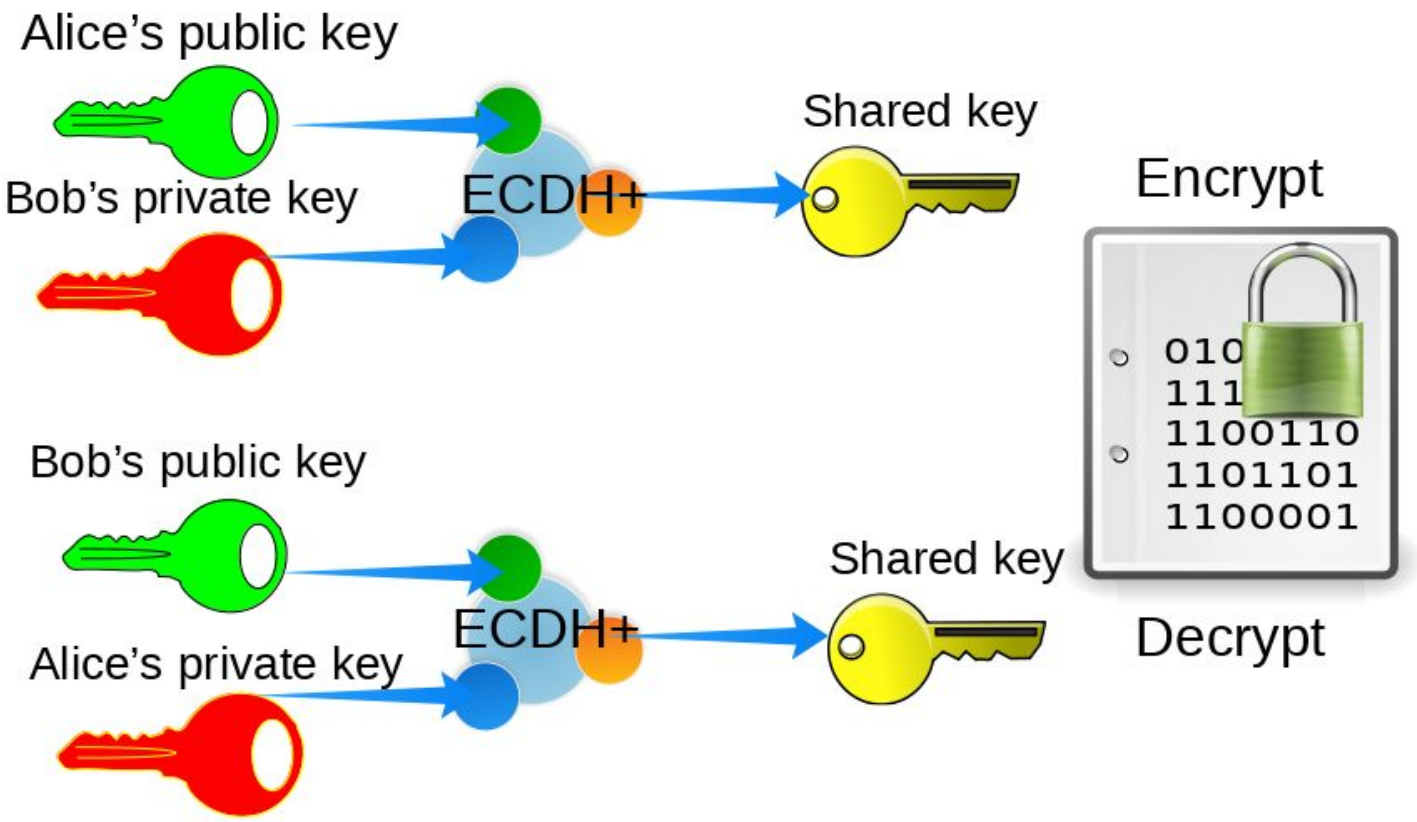
- Just because! :)
- We are experts, but no one expert knows everything!
- 2 Million eyes
- Feedback

Rreading keys and certs

```
X509Certificate test1_cert = KeyReader.readX509CertPEMorDER( new FileInputStream("../
PrivateKey test1_priv = KeyReader.readPrivateKeyPEM(new FileInputStream("../../testda
kpAlice = new KeyPair(KeyReader.extractPublicKeyFromX509(test1_cert), test1_priv);

X509Certificate test2_cert = KeyReader.readX509CertPEMorDER( new FileInputStream("../
PrivateKey test2_priv = KeyReader.readPrivateKeyPEM(new FileInputStream("../../testda
kpBob = new KeyPair(KeyReader.extractPublicKeyFromX509(test2_cert), test2_priv);
```

Shared key explained (Key agreement)



Simple asymmetric encryption (IES)

```
public void testEncryptAsymmetricIES() throws Exception  {
    System.out.println("encryptAsymmetricIES");
    String plain ="Red fox Jumps over Lazy Dog";

    FBCryptoAsym instance1 = new AsymJCEImpl(FBCryptoParams.createDefault());
    instance1.setAsymmetricKeys(kpAlice.getPublic(), kpAlice.getPrivate(), kpBob.getPublic());
    byte[] encrypted = instance1.encryptAsymmetricIES(plain.getBytes());

    FBCryptoAsym instance2 = new AsymJCEImpl(FBCryptoParams.createDefault());
    instance2.setAsymmetricKeys(kpBob.getPublic(), kpBob.getPrivate(), kpAlice.getPublic());
    byte[] decrypted = instance2.decryptAsymmetricIES(encrypted);

    String text = new String(decrypted);
    assertEquals(plain, text);
}
```

Asymmetric encryption with KA

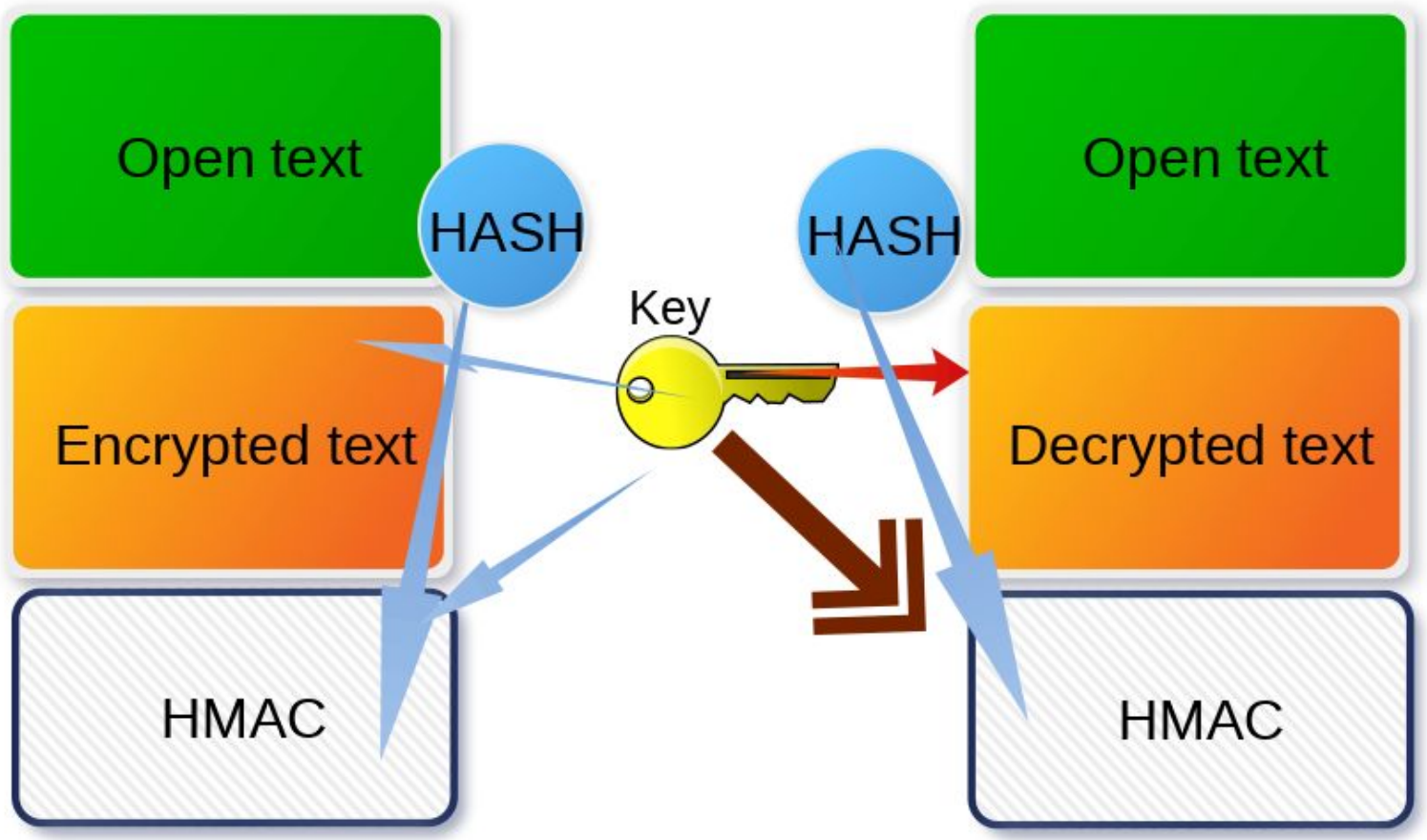
```
public void testEncryptAsymmetric() throws Exception  {
    System.out.println("encryptAsymmetric");
    String plain ="Red fox Jumps over Lazy Dog";

    FBCryptoAsym instance1 = new AsymJCEImpl(FBCryptoParams.createDefault());
    instance1.setAsymmetricKeys(kpAlice.getPublic(), kpAlice.getPrivate(), kpBob.getPublic());
    byte[] encrypted = instance1.encryptAsymmetric(plain.getBytes());

    FBCryptoAsym instance2 = new AsymJCEImpl(FBCryptoParams.createDefault());
    instance2.setAsymmetricKeys(kpBob.getPublic(), kpBob.getPrivate(), kpAlice.getPublic());
    byte[] decrypted = instance2.decryptAsymmetric(encrypted);

    String text = new String(decrypted);
    assertEquals(plain, text);
}
```

What is Authnticated Encrytpion with Associcated Data (AEAD)?



Authenticated encryption with open data

```
public void testEncryptAsymmetricWithAEAData() throws Exception {
    System.out.println("encryptAsymmetricWithAEAData");
    String plain ="Red fox Jumps over Lazy Dog";
    String open ="<<<OPEN TEXT>>>";

    FBCryptoAsym instance1 = new AsymJCEImpl(FBCryptoParams.createDefault());
    instance1.setAsymmetricKeys(kpAlice.getPublic(), kpAlice.getPrivate(), kpBob.getPublic());
    AEADMessage encrypted = instance1.encryptAsymmetricWithAEAData(plain.getBytes(),open.getBytes());

    FBCryptoAsym instance2 = new AsymJCEImpl(FBCryptoParams.createDefault());
    instance2.setAsymmetricKeys(kpBob.getPublic(), kpBob.getPrivate(), kpAlice.getPublic());

    AEAD decrypted = instance2.decryptAsymmetricWithAEAData(encrypted.toBytes());

    String text = new String(decrypted.decrypted);
    String open_r = new String(decrypted.plain);
    String open_e = new String(encrypted.aatext);
    assertEquals(plain, text);
    assertEquals(open, open_r);
    assertEquals(open, open_e);
}
```

Symmetric encryption example

```
public void testEncryptSymmetric() throws Exception {
    System.out.println("encryptSymmetric");
    String plain_text = "Red fox jumps over lazy dog";
    byte[] key = new byte[256/8];
    byte[] salt = new byte[4]; //iv=salt+nounce 12 bytes
    byte[] explicitNounce = new byte[8];
    srand.nextBytes(key);
    srand.nextBytes(salt);
    srand.nextBytes(explicitNounce);
    FBCryptoSym instance1 = new SymJCEImpl(FBCryptoParams.createDefault());
    instance1.setSymmetricSalt(salt);
    instance1.setSymmetricNounce(explicitNounce);
    instance1.setSymmetricKey(key);
    byte[] encrypted = instance1.encryptSymmetric(plain_text.getBytes());

    //in real life we must set salt and key; nounce is prefix of encrypted message
    FBCryptoSym instance2 = new SymJCEImpl(FBCryptoParams.createDefault());
    instance2.setSymmetricKey(key);
    instance2.setSymmetricSalt(salt);
    //ready to decrypt
    byte[] plain = instance2.decryptSymmetric(encrypted);
    String text = new String(plain);
    assertEquals(plain_text, text);
}
```

Basic set of algorytms in FBCrypto

Asymmertic:

- Basic curve: NIST **secp521r1**
- Integrated ECC cipher: **ECIESwithAES-CBC 256 bit**
- Digester: **SHA-512**
- Signature: **SHA512withECDSA**
- AEAD cipher: **AES/GCM/PKCS5Padding**
- AEAD cipher key agreement: **ECDH+SHA-256**(sk,pub1,pub2)

Basic set of algorytms in FBCrypto

Simmetric

- Key length: **256 bit**
- Base cipher: **AES/GCM/PKCS5Padding**
- AEAD auth tag lenght: **128 bit**
- AES IV: **12=4+8 bytes.** 4 is salt, goes with key, 8 is explicit_nounce goes with ciphered data.

Waht is ready & Plans

- Java/BouncyCastle implementaion: Ready, tested. Reference implementation.
- JavaScript: work in progress
- C++/OpenSSL/LibreSSL: work in progress
- Hardware backed: work in progress
- Other languages? Help is needed!

Source Code

- GIT repository at:
- https://bitbucket.org/firstbridge_company/fb-crypto-public
- License: GPL v.2
- We are open, join us!

Authors, contacts

- FirstBridge company: http://firstbridge.io
- Oleksiy Lukin (Java,C++): alukin@gmail.com
- Leonid Strizhevskiy(Java):
- Hryhorii Chaikovskyi(JavaScript):