Hardcore Crypto: A look at the state of cryptography

David Hook

VP of Crypto Workshop

Keyfactor

What's Going On:
In the headlines of cryptography

NIST Lightweight Cryptography

The NIST competition entering it's final stage.

Post Quantum Cryptography

Two algorithms now standardized, 4 scheduled for standardization. There's fine print on patents and also more algorithm "adjustments" coming.

Bouncy Castle progress

Crypto Workshop is submitting a hybrid Java module for FIPS 140-3.

Certification requests, not as we know them

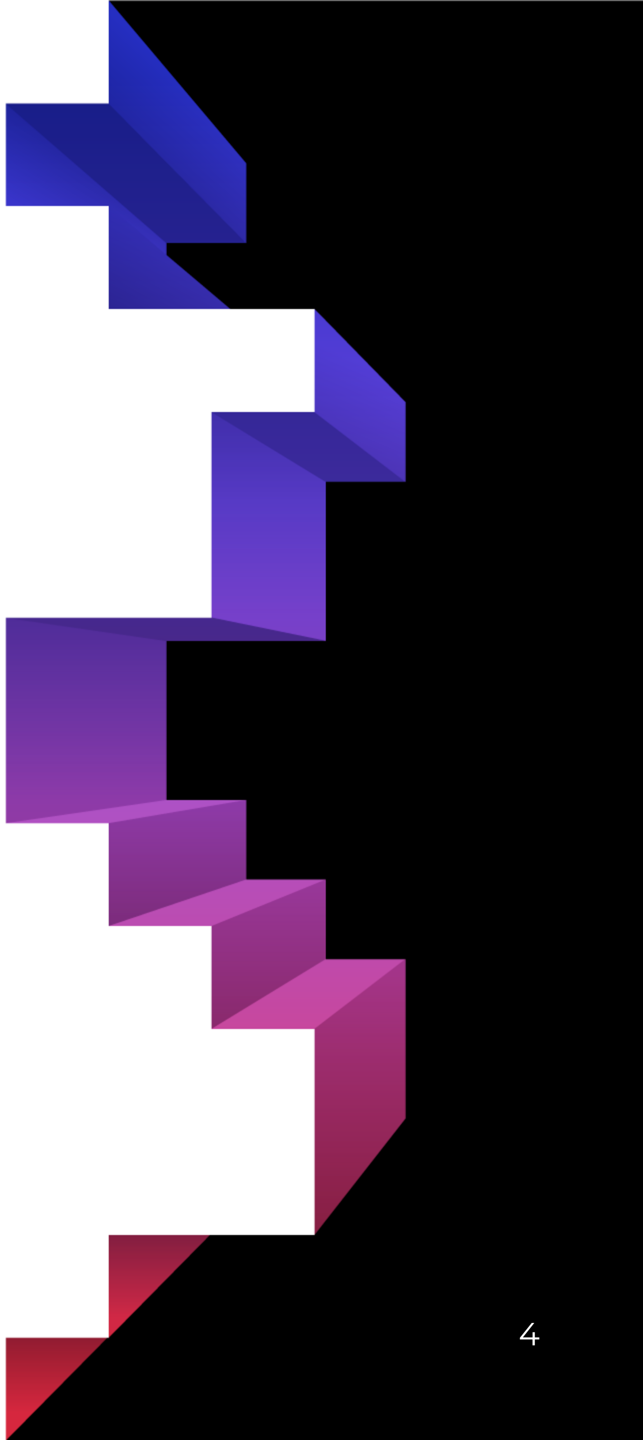
PQC Key Encapsulation Mechanisms (KEM) algorithms cannot be used to create PKCS#10 style certification requests. CMP/CRMF "encrCert" protocol is suddenly a real thing.

Lightweight Cryptography

Security for constrained environments

NIST Lightweight Cryptography

- An effort that started with an initial study in 2013, now finally resulting in some algorithms.
- Task was to find algorithms that would work effectively on small devices with limited hardware, including where software-based algorithms were not an option.
- Initial call went out in 2018, with 56 submissions received.
- Algorithms also looked at from a perspective of side channel protection as well as performance.



Competition Status: Lightweight Cryptography

Now we're down to 10 finalists from the initial field of 56 in 2018.

All algorithms offer AEAD modes. Some algorithm families include message digests.

Grain128AEAD was included in BC Java 1.72 and in BC C# .NET 2.0.0

The rest of the candidates now in process for addition to the BC APIs.

Final selection to be announce "any day now".

Post-Quantum Cryptography

What’s happening, where it’s heading

PQC Status

- The two signature algorithms LMS and XMSS described in SP 800-208 are still the only ones standardized.
- The signature algorithms Dilithium, Falcon, SPHINCS+, and KEM algorithm Kyber are scheduled for standardization.
- KEMs (Round 4) Classic McEliece, BIKE, and HQC – these are all code based algorithms.
- Proposals for additional PQC signature algorithms now been accepted as well.

PQC: What to watch out for

- Final parameter sets are still being determined and some changes have also been proposed.
- Patents for some algorithms are still problematic.
- It is not just NIST producing standards.
- Some proposed changes will create a subset of algorithms which are more “lightweight”.

PQC: Why Patents Still Matter

- Using KEMs to implement Hybrid Cryptography is an obvious choice for protecting against encrypted data harvesting.
- The preferred algorithm Kyber has patent exposure both in the US and France.
- NIST IP Statement for Kyber appears to only talk about the final algorithm as standardized. This is not necessarily the same algorithm we are looking at now.
- Fortunately NTRU (the round 3 candidate) is patent free, with the NTRU-HRS parameter set already in use.

PQC: Summary

- Start experimenting.
- Think about your data – explore the use of hybrid encryption for transmitting anything you want protected long-term.
- For long-life code signing, use LMS or XMSS, or wait till mid-2024.
- All the round 3 candidates and finalists are now available in the Bouncy Castle APIs, both Java and C#.

The Bouncy Castle APIs

FIPS, auxiliary libraries, and further work

What's coming in FIPS

- Currently focused on having both Java and C# FIPS 140-3 certified modules.
- Java FIPS 140-3 is already in coordination.
- C# FIPS 140-3 is to be submitted in the next month or two.
- Additional updates for the end of transition for RSA PKCS#1.5 and the use of TDES encryption.
- BC-FJA 2.1.0 will also provide access to AES and DBRG hardware on Intel Operational Environments.

Bouncy Castle APIs

- Further support for the NIST lightweight cryptography candidates.
- KEM support for CMS, CRMF.
- Message Layer Security (now at Draft 17).
- Continued work on hybrid certificates, both as defined in current draft RFCs and section 7.2.2 of X.509 (201910).

Certification Requests

A study in change

So, what's all this about KEMs?

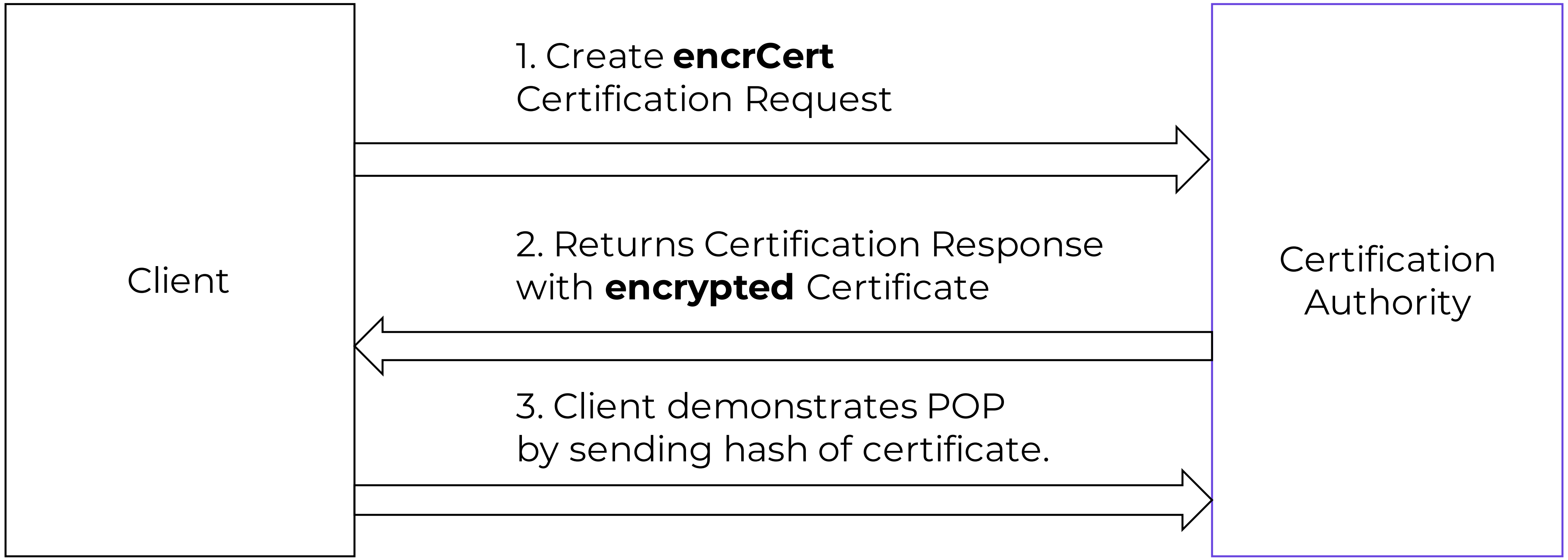
1 KEMs cannot be used for generating signatures.

2 So other than 3rd party signing the “PKCS#10 party” is over.

3 CRMF (RFC 4211) is actually designed to deal with this but it requires an extra acknowledgment step.

4 The approach is known by its subsequent Message flag value “encrCert”

Getting a Certificate for a KEM


Client
1. Create **encrCert**
Certification Request
2. Returns Certification Response
with **encrypted** Certificate
3. Client demonstrates POP
by sending hash of certificate.
Certification
Authority

KEM CRMF – Request Generation

```
KeyPair ntKp = … // NTRU key pair

JcaCertificateRequestMessageBuilder certReqBuild =
               new JcaCertificateRequestMessageBuilder(BigIntegers.ONE)
         .setPublicKey(ntKp.getPublic())
         .setSubject(X500Name.getInstance(sender.getName()))
         .setProofOfPossessionSubsequentMessage(SubsequentMessage.encrCert);

CertificateReqMessagesBuilder certReqMsgsBldr =
                new CertificateReqMessagesBuilder().addRequest(certReqBuild.build());

MacCalculator senderMacCalculator =
      new JcePBMac1CalculatorBuilder("HmacSHA256", 256)
           .setProvider("BC").build(senderMacPassword);

ProtectedPKIMessage message = new ProtectedPKIMessageBuilder(sender, recipient)
        .setBody(PKIBody.TYPE_INIT_REQ, certReqMsgsBldr.build())
        .build(senderMacCalculator);
```

KEM CRMF – Generate Encrypted Response

```
CMSEnvelopedDataGenerator edGen = new CMSEnvelopedDataGenerator();

// note: use cert req ID as key ID, don't want to use issuer/serial in this case!
edGen.addRecipientInfoGenerator(new JceKeyTransRecipientInfoGenerator(
            senderReqMessage.getCertReqId().getEncoded(),
            new JceAsymmetricKeyWrapper(new KEMParameterSpec("AES-KWP", 192),
            new
JcaX509CertificateConverter().setProvider("BC").getCertificate(cert).getPublicKey())));

CMSEnvelopedData encryptedCert = edGen.generate(
                        new CMSProcessableCMPCertificate(cert), new
JceCMSContentEncryptorBuilder(CMSAlgorithm.AES192_CBC).setProvider("BC").build());

CertificateResponseBuilder certRespBuilder = new CertificateResponseBuilder(
                                      senderReqMessage.getCertReqId(), new
PKIStatusInfo(PKIStatus.granted));
certRespBuilder.withCertificate(encryptedCert);
```

KEM CRMF – Generate Confirmation

```
// extract certificate
CMPCertificate receivedCMPCert = certResp.getCertificate(
                     new JceKeyTransEnvelopedRecipient(ntKp.getPrivate()));

X509CertificateHolder receivedCert =
                new X509CertificateHolder(receivedCMPCert.getX509v3PKCert());

// confirmation message calculation
CertificateConfirmationContent content =
     new CertificateConfirmationContentBuilder()
          .addAcceptedCertificate(receivedCert, BigInteger.ONE)
          .build(new JcaDigestCalculatorProviderBuilder().build());

message = new ProtectedPKIMessageBuilder(sender, recipient)
    .setBody(PKIBody.TYPE_CERT_CONFIRM, content).build(senderMacCalculator);
```

Resources

NIST Lightweight Cryptography

https://csrc.nist.gov/Projects/Lightweight-Cryptography

NIST PQC Cryptography Competition

https://csrc.nist.gov/projects/post-quantum-cryptography

For the full CRMF CMP example see

https://github.com/bcgit/bc-java/blob/master/pkix/src/test/java/org/bouncycastle/cert/cmp/test/PQCTest.java

Everything about the Bouncy Castle APIs

https://www.bouncycastle.org

Thank you!
Questions?

David Hook

VP, Crypto Workshop

Keyfactor