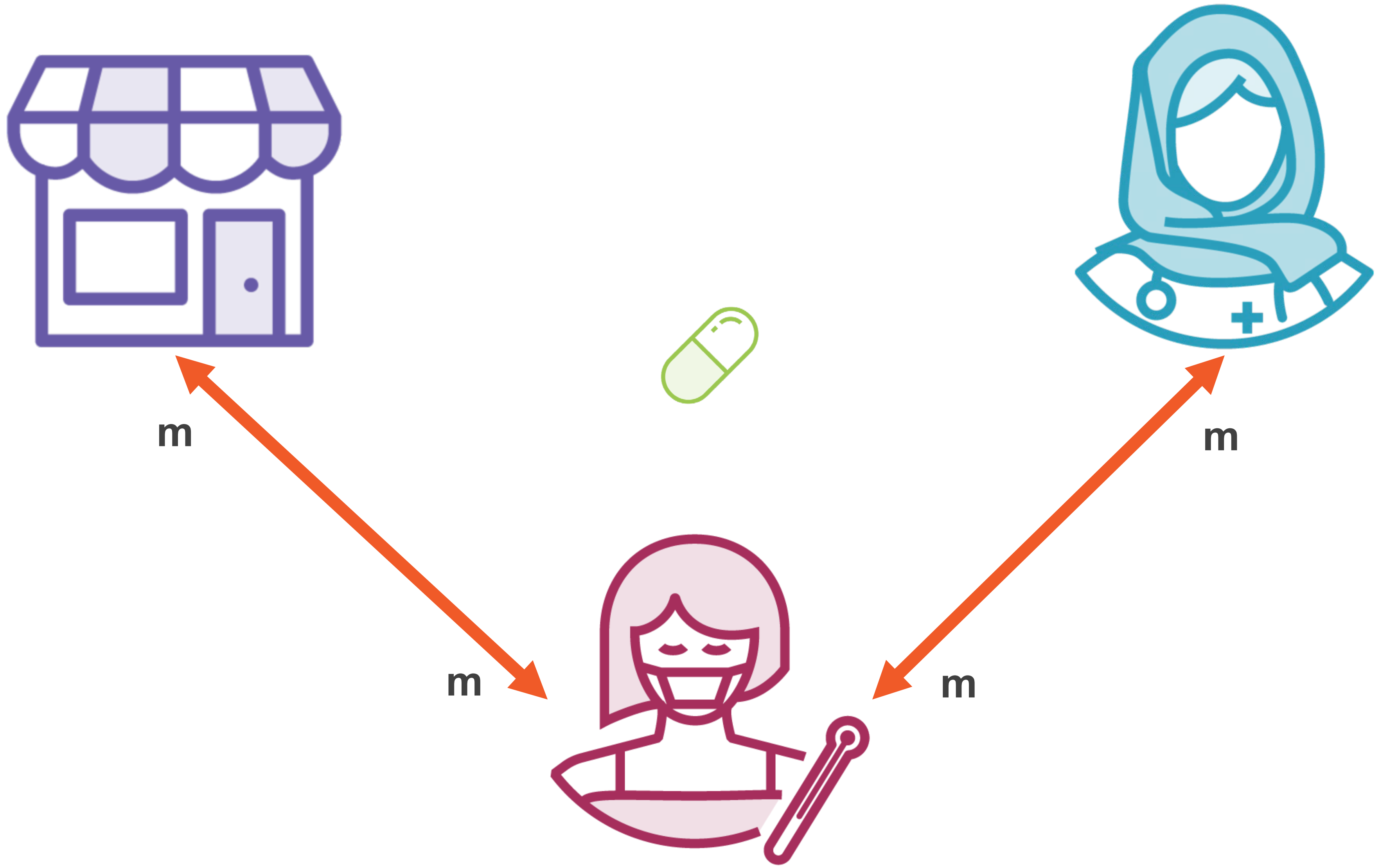
Securely Signing and Verifying Data

Josh Cummings
PRINCIPAL SOFTWARE ENGINEER

@jzheaux blog.jzheaux.io

Distributed Medicine



What If a Prescription Is…

Blank?	**Stolen?**	**Copied?**	**Altered?**
Identifiers	Patient Name	Special Paper	Special Encoding

Encoding isn’t security

What If a Prescription Is…

Blank?	**Stolen?**	**Copied?**	**Altered?**
DEA Number	Patient Name	Special Paper	Special Encoding
Authenticity	Authenticity	Replayability	Integrity

Demo

Bank Transfer Demo

What If a Prescription Is...

Blank?

DEA Number

Authenticity

Stolen?

Patient Name

Authenticity

Altered?

Special Encoding

Integrity

Parity

From: Dr. Watson
For: Hacker Doe

30 cc. of Penicillin
819

From: Dr. Watson
For: Hacker Doe

80 cc. of Penicillin
819

30 x 27.3 = **819**

80 x 27.3 = **2184**

Mac Integrity Checks

Sender

```
Mac mac =
Mac.getInstance(“HMACSHA256”);
mac.init(secretKey);

mac.update(clientId);
mac.update(accountNumber);
mac.update(amount);

byte[] senderSignature =
mac.doFinal();

send(clientId, accountNumber,
          amount, senderSignature);
```

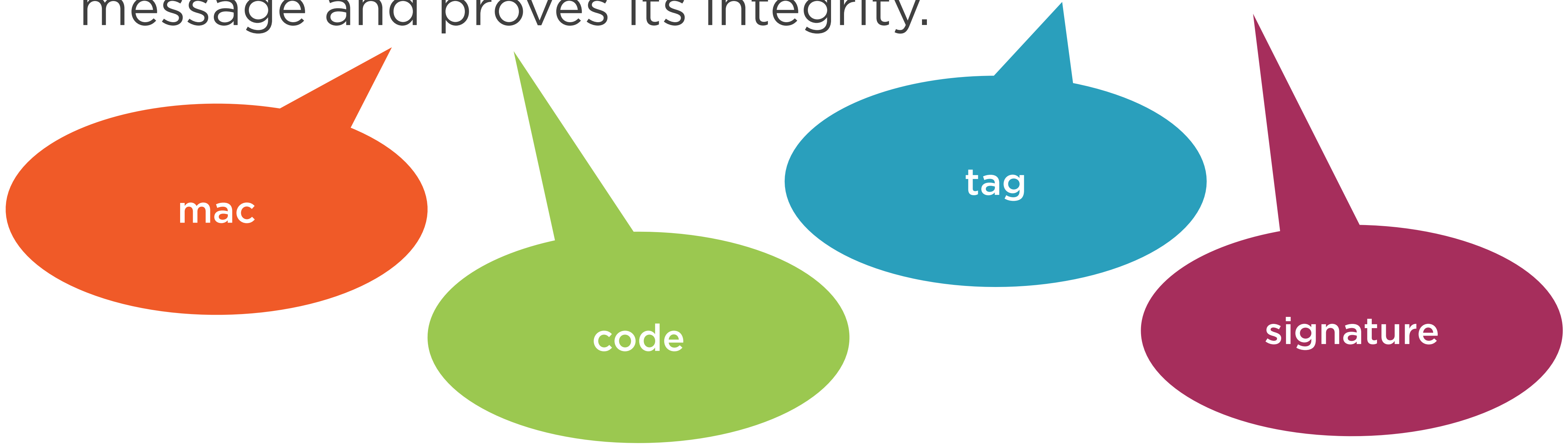
Receiver

```
Mac mac =
Mac.getInstance(“HMACSHA256”);
mac.init(secretKey);

mac.update(clientId);
mac.update(accountNumber);
mac.update(amount);

byte[] receiverSignature =
mac.doFinal();

if (senderSignature ==
        receiverSignature)
    // hooray!
```

```
= hmac(secret, message)
```

Message Authentication Code

A short piece of information that authenticates a message and proves its integrity.



```
Mac mac = Mac.getInstance("HMACSHA256")
mac.init(secretKey);

mac.update(each);
mac.update(individual);
mac.update(property);

byte[] signature = mac.doFinal();
```

Mac

Use `getInstance`, like other JCA classes, and give it a symmetric key

Everything contributes to the signature

Make sure to encode the signature for easy transport

All message data contributes to the signature

Demo

Mac

Timing Attacks

Blind SQL Injection

Invalid queries return faster than valid ones

Authentication Enumeration

Invalid usernames fail authentication faster than valid usernames

Timing Attacks

0000000000000000?

No

1100000000000000?

No

2200000000000000?

Ahh.. no

3300000000000000?

No

Demo

Mac

So, Why Not Just Use MessageDigest?

`MessageDigest`	`Mac`
hash(message)	*hash*(secret + message)
Anyone can produce	**Only the sender or receiver can produce**
Integrity Only	**Authenticity + Integrity**

Non-repudiation

A characteristic of a signature by which a sender cannot at a later date deny having signed it.

Digital Signature Integrity Checks

Sender

```
Signature signature =
        getInstance("SHA256RSA");
signature.initSign(privateKey);

signature.update(bankId);
signature.update(accountNumber);
signature.update(amount);

byte[] sender =
        signature.sign();

send(bankId, accountNumber,
          amount, sender);
```

Receiver

```
Signature signature =
        getInstance("SHA256RSA");
signature.initVerify(publicKey);

signature.update(bankId);
signature.update(accountNumber);
signature.update(amount);

boolean verified =
        signature.verify(sender);

if (verified)
    // hooray!
```

```
Signature signature = Signature.getInstance("SHA256WITHRSA")
signature.init(secretKey);

signature.update(each);
signature.update(individual);
signature.update(property);

byte[] signature = signature.sign();
```

Signature

Use `getInstance`, like other JCA classes, and give it a private key to sign

Everything contributes to the signature

Make sure to encode the signature for easy transport

Demo

Signature

Downgrade Attacks

Declare Expectations

```
if (alg == "v1")
then verify()
```

Exclude Weak Algorithms

```
if (alg == "none")
then throw
```

Key Confusion

```
else throw
```

What If a Prescription Is...

Copied?

Replayability

Replay Attack

When an attacker can induce a recipient to accept and process the same message more than once.

Include a Guid

```
{
    “id” : “1234-abcd-5687-efab”,
    “version” : “v1”,
    “signature” : “23BEDOB=49QWEQWWE/”,
    “clientId” : “92834233”,
    “accountNumber” : “987654321”,
    “amount” : “92.00”
}
```

Smell Test

Include Timestamps

```
{
    “id” : “1234-abcd-5687-efab”,
    “created” : “2019-01-03T12:23:34”,
    “version” : “v1”,
    “signature” : “23BEDOB=49QWEQWWE/”,
    “clientId” : “92834233”,
    “accountNumber” : “987654321”,
    “amount” : “92.00”
}
```

Demo

Nonces

Key Sharing Options

Out-of-Band

Give the key in a way that is not part of the message handshake

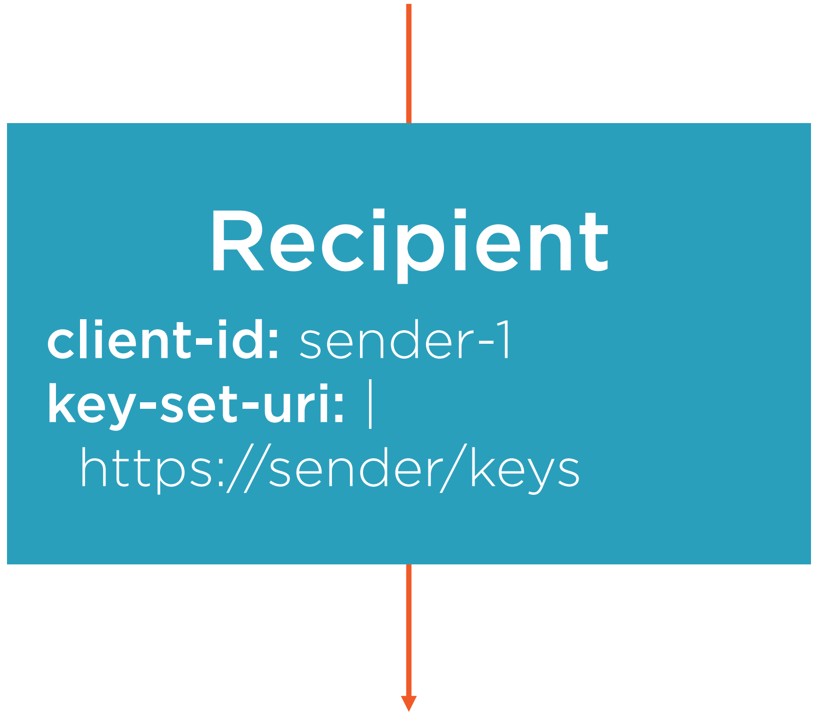
Key Service Endpoint

Sender serves the public keys from a known endpoint

Serving Public Keys

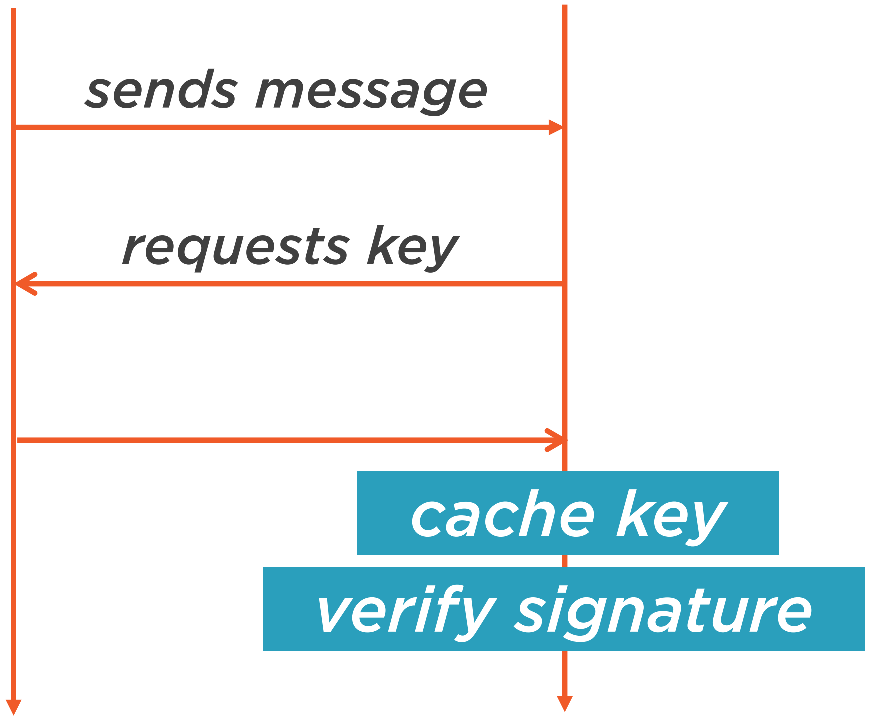
1 **Register the endpoint**

2 **Check endpoint in handshake**



Recipient

client-id: sender-1
key-set-uri: |
  https://sender/keys



JWS: A Protocol for Signing JSON

This Module	JWS
signature version	alg
message id	jti
creation date	iat
sender id	iss
key id	kid

```
NimbusJwtDecoder jwtDecoder =
    NimbusJwtDecoder.withJwkSetUri
        (“https://recipient/keys”).build();
Jwt jwt = jwtDecoder.decode(messageBody);
Map<String, Object> messageData = jwt.getClaims();
```

JWS Support in Spring Security 5.x

Based on the Nimbus library

The class is called `NimbusJwtDecoderJwkSupport` **in 5.1, with the more powerful** `NimbusJwtDecoder` **coming in 5.2**

Demo

Nimbus

(De)serialization

Several aspects of a message can be faked, altered, or copied

MACs and Digital Signatures are ways to establish integrity and authenticity

All data goes into a signature

Nonces are helpful for replay attacks

Key service endpoints for Key Rotation

JWS, WS-Security and other protocols exist – Spring Security and several other libraries offer JWS support