# Understanding the Cryptographic Principles used with Blockchain

**Stephen Haunts**

LEADER, DEVELOPER, SPEAKER AND TRAINER

@stephenhaunts www.stephenhaunts.com

## Summary

Hashing

Authenticated hashing (HMAC)

Digital signatures

Practical Cryptography in .NET

By Stephen Haunts

https://app.pluralsight.com/library/courses/practical-cryptography-dotnet

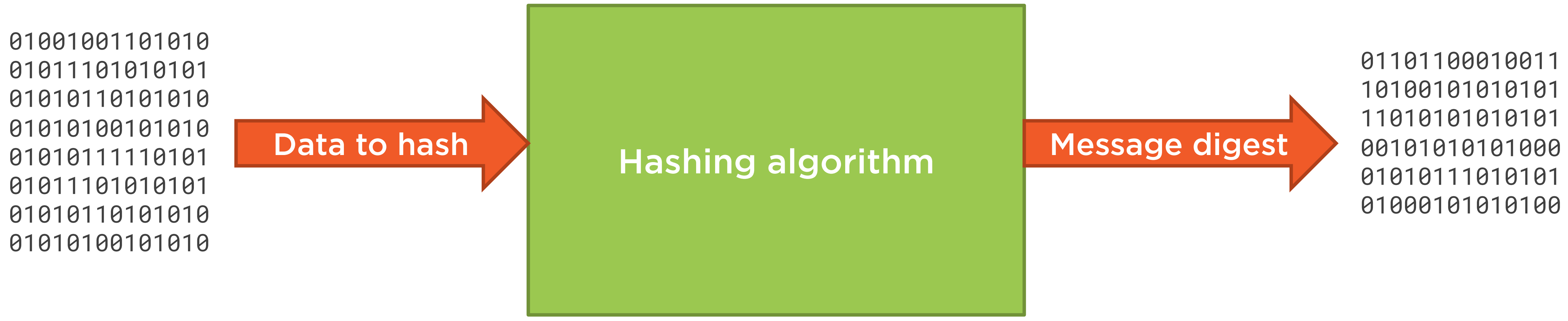
## Play by Play : Enterprise Data Encryption in Azure Revealed

By Stephen Haunts

https://app.pluralsight.com/library/courses/play-by-play-enterprise-data-encryption-with-azure-revealed

## Hashing

# Hashing


01001001101010
01011101010101
01010110101010
01010100101010
01010111110101
01011101010101
01010110101010
01010100101010
Data to hash
Hashing algorithm
Message digest
01101100010011
10100101010101
11010101010101
00101010101000
01010111010101
01000101010100

## Hashing

Easy to compute the hash value

Infeasible to generate a message that has a given hash

Infeasible to modify a message without changing the hash

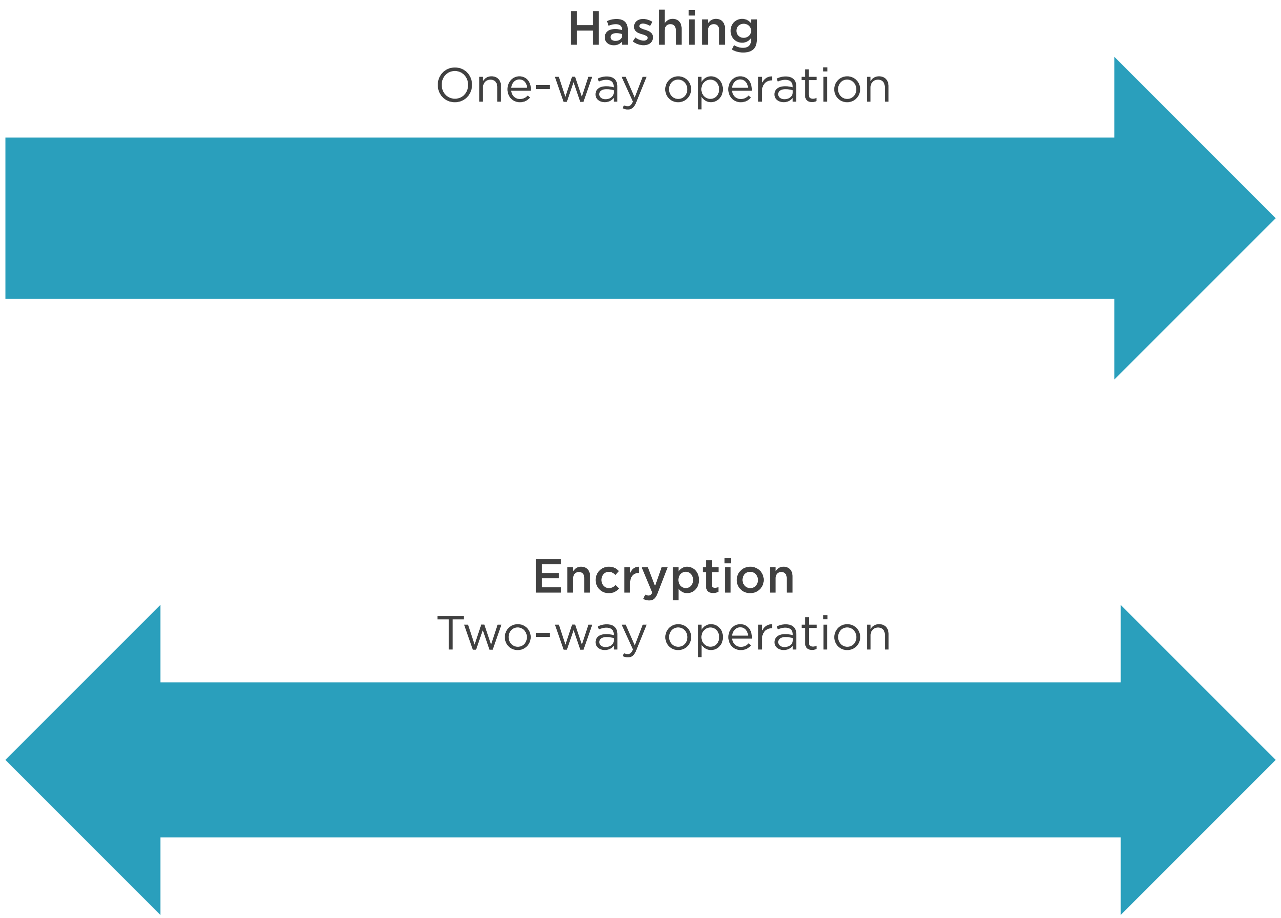
Infeasible to find two different messages with the same hash

MD5

SHA-1

SHA-256

SHA-512


**Hashing**
One-way operation
**Encryption**
Two-way operation

MD5

SHA-1

**SHA-256**

SHA-512

## Secure Hash Algorithm (SHA) Family

SHA-1

SHA-2

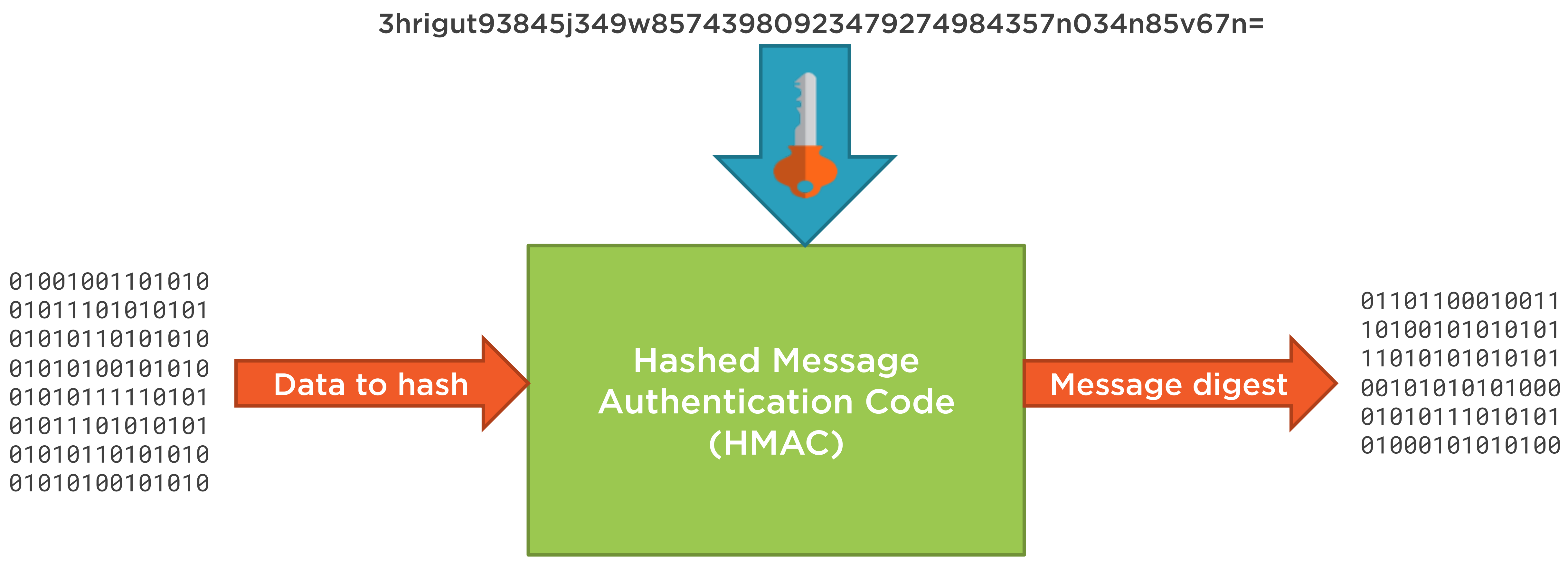
SHA-3

## Secure Hash Algorithm (SHA) Family

```
public static byte[] ComputeHashSha256(byte[] toBeHashed)
{
    using (var sha256 = SHA256.Create())
    {
        return sha256.ComputeHash(toBeHashed);
    }
}
```

## Authenticated Hashing (HMAC)

# Hashed Message Authentication Codes



## Hashed Message Authentication Codes

$$\left.\begin{array}{c}\textbf{Integrity}\\ \textbf{+}\\ \textbf{Authentication}\end{array}\right\}\ \textbf{HMAC}$$

## Hashed Message Authentication Codes

```
public static string GetHash(string toBeHashed, string key)
{
    var keyToUse = Encoding.UTF8.GetBytes(key);
    var message = Encoding.UTF8.GetBytes(toBeHashed);
    using (var hmac = new HMACSHA256(keyToUse))
    {
        return Convert.ToBase64String(hmac.ComputeHash(message));
    }
}
```

## Digital Signatures

## Digital Signatures

The sender can not deny sending the message

## Digital Signatures

$$\left.\begin{array}{c}\textbf{Authentication}\\ +\\ \textbf{Non-repudiation}\end{array}\right\}\ \textbf{Digital signature}$$

Created by Freepik

## Digital Signatures

Public and private key generation

Signing algorithm

Signature verification algorithm

Practical Cryptography in .NET

By Stephen Haunts

https://app.pluralsight.com/library/courses/practical-cryptography-dotnet

## Digital Signatures

## Digital Signatures

1. **Alice encrypts her data**

# Digital Signatures

1. Alice encrypts her data
2. Alice takes a hash of her data

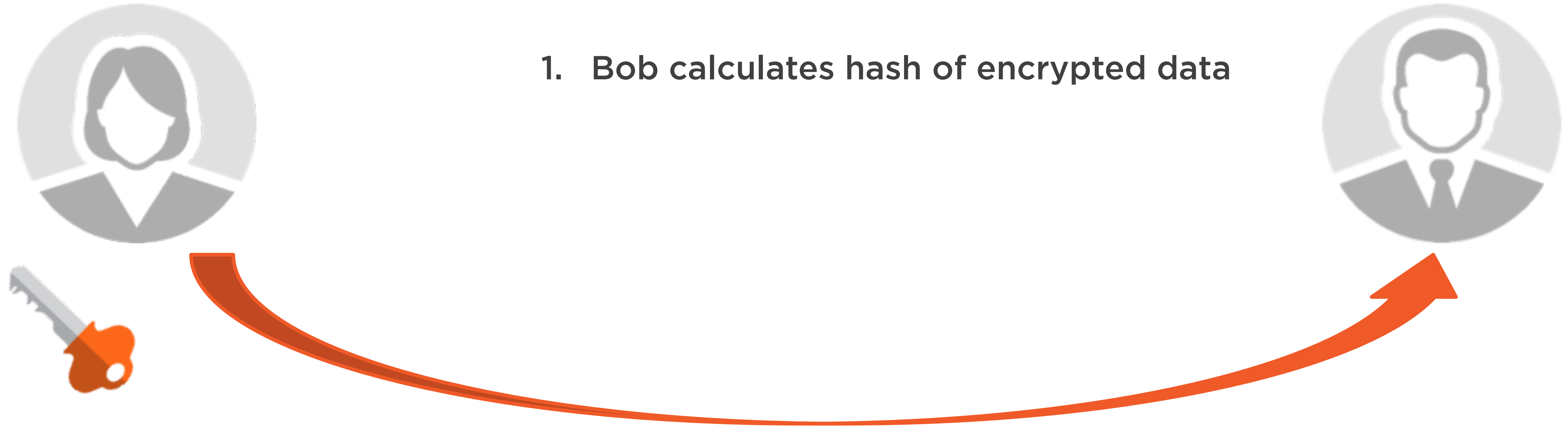
# Digital Signatures

1. Alice encrypts her data
2. Alice takes a hash of her data
3. Alice signs the data with her private signing key

## Digital Signatures

Alice sends encrypted data, the hash and the digital signature to Bob

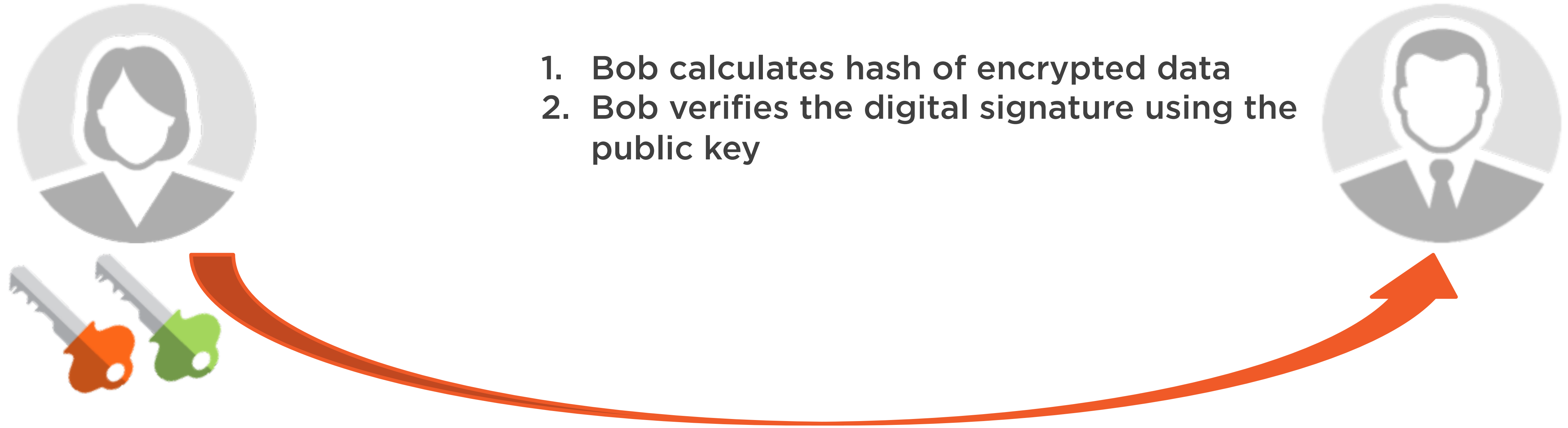
# Digital Signatures



1. **Bob calculates hash of encrypted data**

Alice sends encrypted data, the hash and the digital signature to Bob

# Digital Signatures



1. Bob calculates hash of encrypted data
2. Bob verifies the digital signature using the public key

Alice sends encrypted data, the hash and the digital signature to Bob

## Digital Signatures

| | Public Key | Private Key |
|---|---|---|
| Encryption (RSA) | | |
| Digital signatures | | |

## Digital Signatures

| | Public Key | Private Key |
|---|---|---|
| Encryption (RSA) | Encrypt | Decrypt |
| Digital signatures | | |

# Digital Signatures

| | Public Key | Private Key |
|---|---|---|
| Encryption (RSA) | Encrypt | Decrypt |
| Digital signatures | Verify signature | Sign message |

# Digital Signatures

```
private RSAParameters _publicKey;
private RSAParameters _privateKey;


public void AssignNewKey() {
    using (var rsa = new RSACryptoServiceProvider(2048)) {
        rsa.PersistKeyInCsp = false;


        _publicKey = rsa.ExportParameters(false);
        _privateKey = rsa.ExportParameters(true);
    }
}
```

## Digital Signatures

```
public byte[] SignData(byte[] hashOfDataToSign) {
    using (var rsa = new RSACryptoServiceProvider(2048)) {
            rsa.PersistKeyInCsp = false;
            rsa.ImportParameters(_privateKey);

            var rsaFormatter = new RSAPKCS1SignatureFormatter(rsa);
            rsaFormatter.SetHashAlgorithm("SHA256");

            return rsaFormatter.CreateSignature(hashOfDataToSign);
        }
 }
```

## Digital Signatures

```
public bool VerifySignature(byte[] hashOfDataToSign, byte[] signature) {
    using (var rsa = new RSACryptoServiceProvider(2048)) {
        rsa.ImportParameters(_publicKey);

        var rsaDeformatter = new RSAPKCS1SignatureDeformatter(rsa);
        rsaDeformatter.SetHashAlgorithm("SHA256");

        return  rsaDeformatter.VerifySignature(hashOfDataToSign, signature);
    }
}
```

## Summary

Hashing

**SHA-256**

Authenticated hashing (HMAC)

**SHA-256**

Digital signatures

**RSAPKCS1-SignatureFormater**