Asymmetric Cryptography Primer

Presenter: MJ Kwon – Senior Embedded Solutions Engineer

MICROCHIP

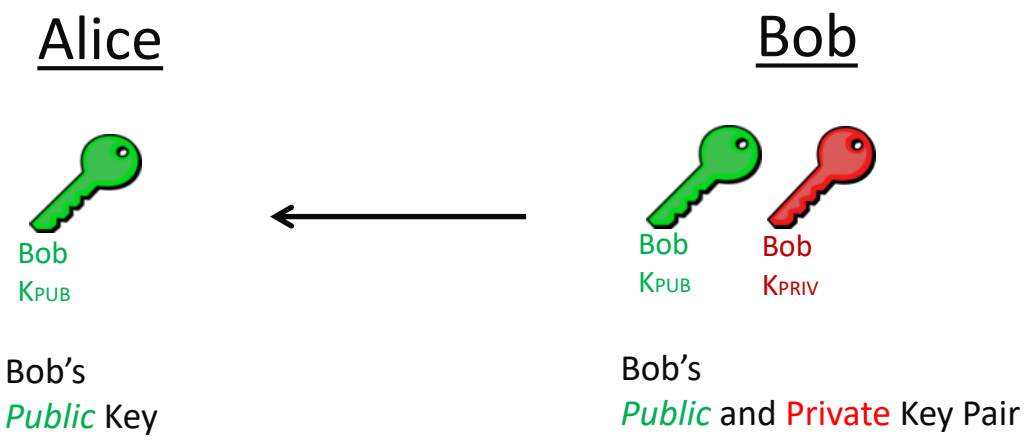
Public Key Cryptography

"Asymmetric", "Public/Private Key" or "Public Key Infrastructure (PKI)" Cryptography

- **Uses two *mathematically related* keys**
- **Imagine having two *different keys* to the same lock box**
  - If the *first key* locked it, only the *second key* can unlock it
- **Key exchange becomes easy**
  - One key can be shared publicly and is called the "**Public Key**"
  - The other is held *very privately* and is called the "**Private Key**"
    - The security and privacy of this key is critical to the security of any asymmetric system
- ***In the Public Key Infrastructure (PKI), every element has its own key pair***
  - Key pairs should ***never*** be shared
    - The basis for security is the unique identity of each element

This is Key Transport

- **Bob directly provides his public key to Alice**
  - Alice and Bob know and trust each other
  - Alice is assured it is genuine and will trust it based on her knowledge and trust in Bob

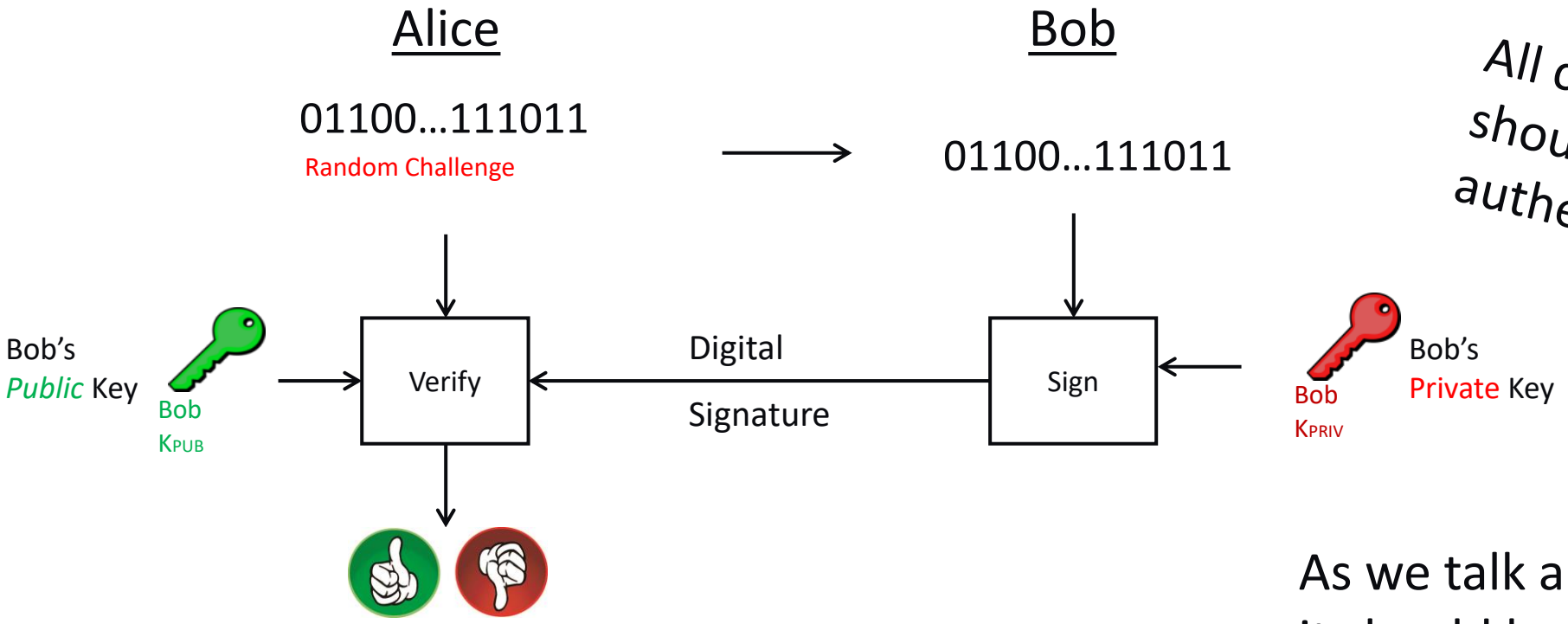


Alice can provide Bob her public key in the same way.
They know and trust each other

MICROCHIP

Asymmetric Authentication Example

- **Alice wants to authenticate she is connected to Bob**
  Alice and Bob know and trust each other and have their own public/private key pairs

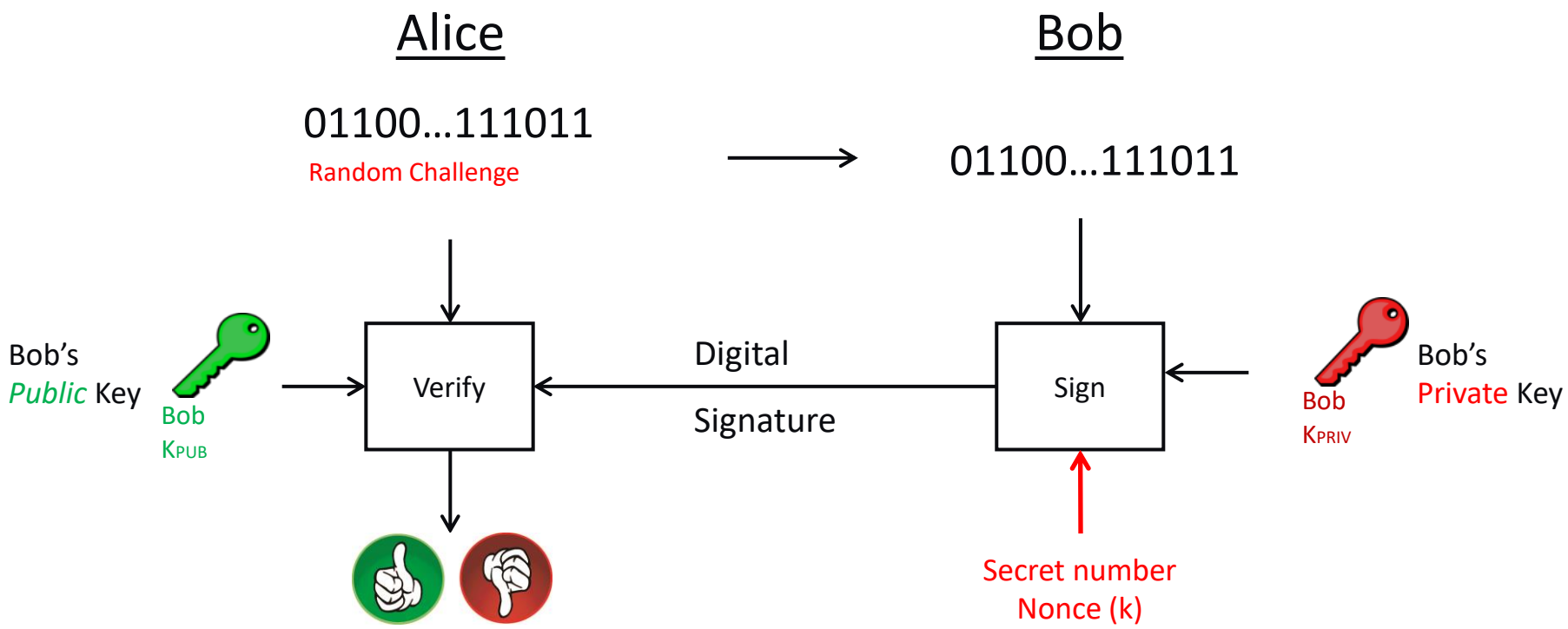


All communications should start with an authentication!

We can imagine how this method could also be used to authenticate a disposable or accessory.

As we talk about other functions, it should be assumed that any communication first started with an authentication!

More Accurately, a Signing Looks Like This

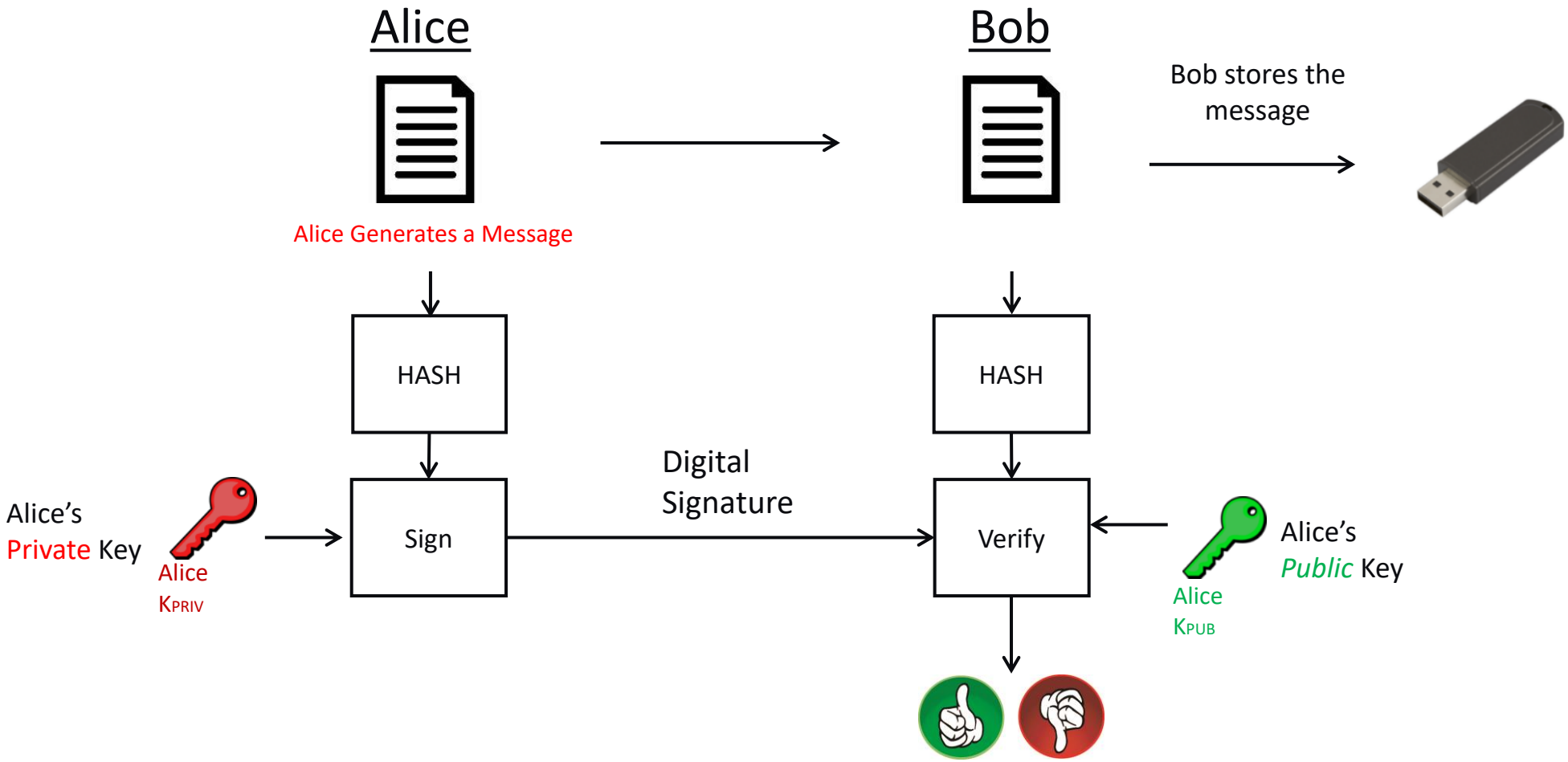


A "known-plaintext attack" is an attempt to calculate secrets from output produced from known input.

Injecting randomness defeats this attack. Even if the *same exact challenge* is signed, a *completely different, yet verifiable, signature* is calculated.

Asymmetric Message Authentication

- **Alice wants to send a message to Bob**
  Alice and Bob know and trust each other and have their own public/private key pairs



MICROCHIP

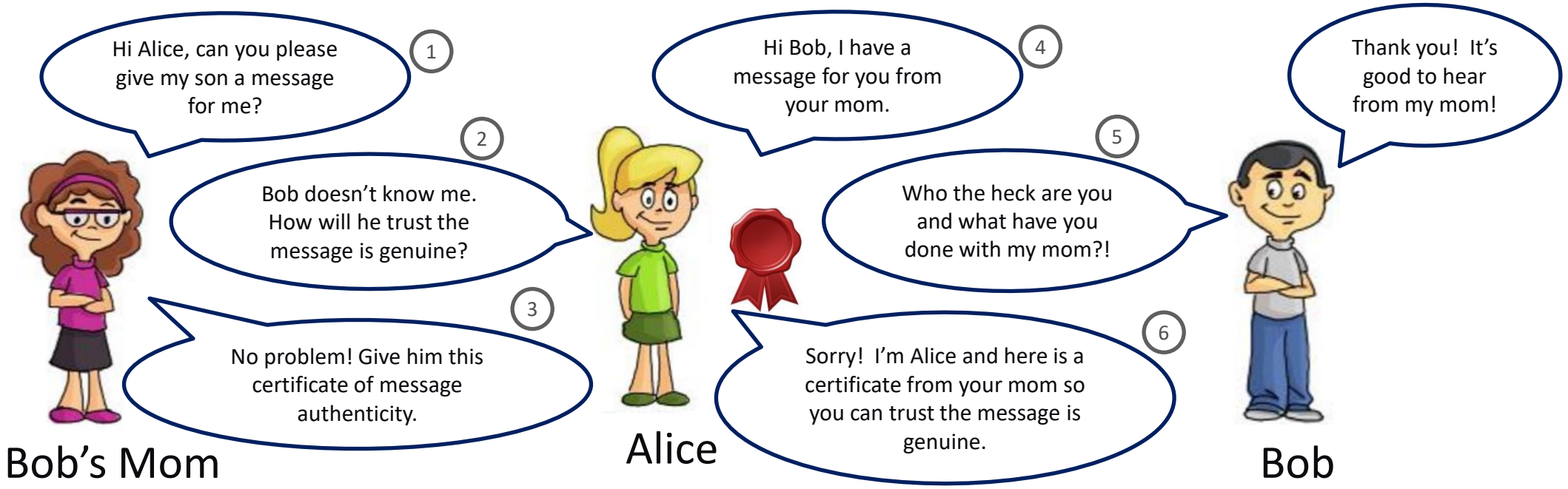
What if Alice and Bob Don't Know Each Other?

- **We've only considered the case of Alice and Bob knowing and trusting each other**
  - If Alice and Bob are strangers, they must establish a *chain of trust* in which they share a common *authority* or *trust anchor* or *root of trust*
  - This trusted *authority* will essentially ***vouch for*** Alice to Bob and will ***vouch for*** Bob to Alice
- **Certificates are used to carry the authority's assurance that these entities are genuine**
  - The authority is typically the OEM in the embedded space, but can be a Certificate Authority (CA) like:
    - Symantec®
    - Go Daddy®
    - Digicert®
    - GlobalSign®
    - Kyrio®
    - Let's Encrypt®
    - Etc.

How are Certificates Used to Establish Trust?

Cartoon Example Using Bob's Mom as the *Root of Trust*

Alice knows and trusts Bob's mom but has never met Bob



This is an example of *message authentication*. But you could imagine Bob's mom could have given Alice a certificate directing Bob to trust whatever Alice says, essentially making the introduction and removing herself. That would be a way to *transfer authority*.

MICROCHIP

Signatures and the Structure of a Certificate

Accepted PKI Standard is the X.509 v3 Digital Certificate

Certificate Content

- Version and Serial Number
- Algorithm ID
- Issuer
- Validity
  - Not Before / Not After
- Subject
- Subject Public Key Info
  - Public Key Algorithm
  - Subject Public Key
- Certificate Signature Algorithm
  - Digital Signature
- Issuer Unique Identifier (optional)
- Subject Unique Identifier (optional)
- Extensions (optional)

Alice's Public Key

Signature from Bob's mom

See the following link for Certificate details:

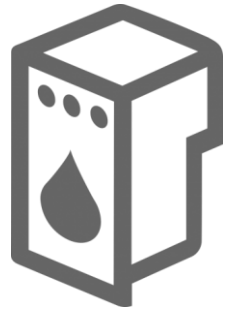
https://tools.ietf.org/html/rfc5280

Alice gave Bob's mom this document and requested she sign it (CSR – certificate signing request).

Bob's mom hashed the document, including Alice's signature, and signed the digest, thereby assuring all the information contained in it.

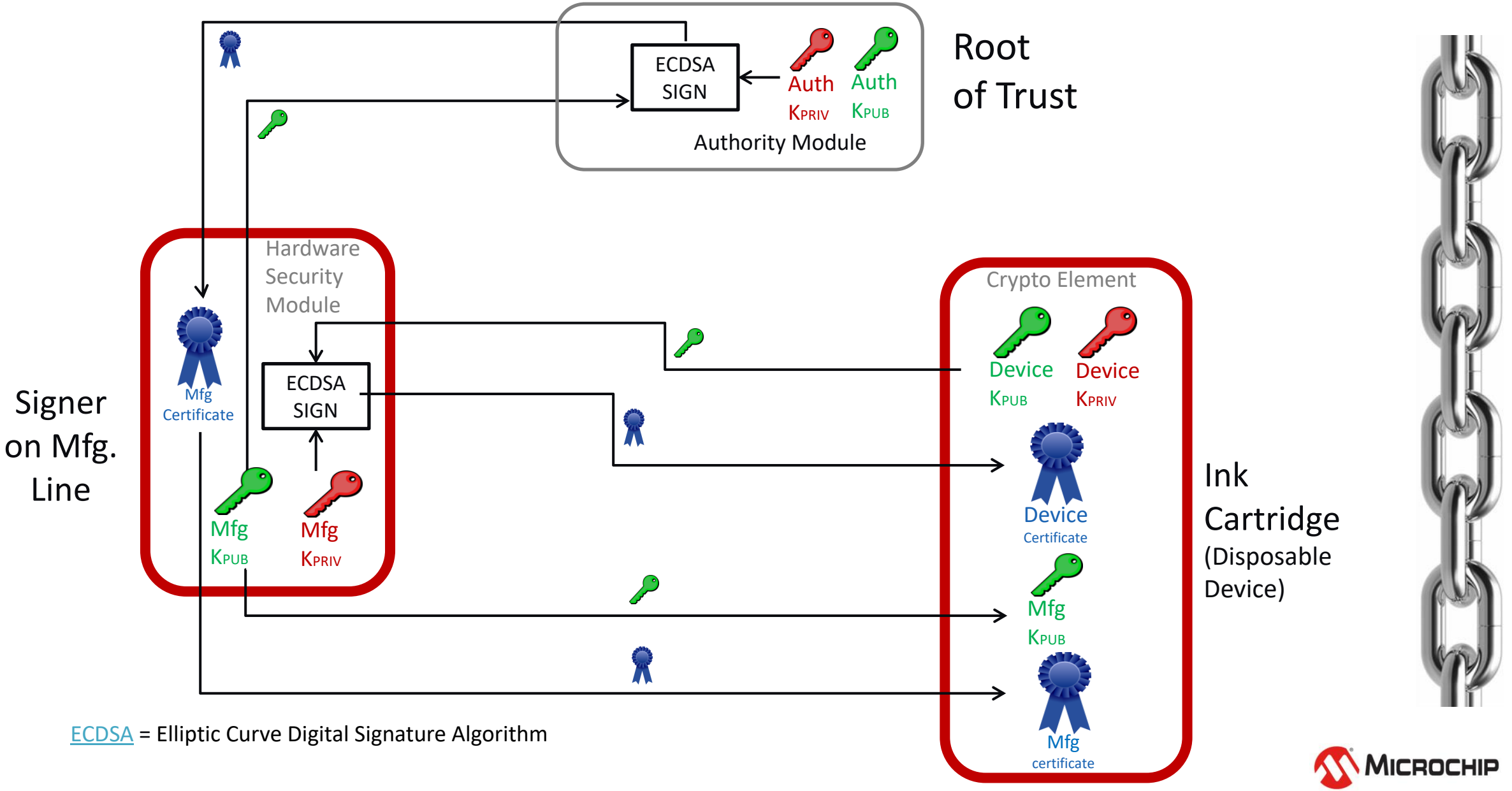
How Certificates (Digitally Signing) are Used in a Real-World System

Asymmetrically authenticating ink cartridges, for example



Ink cartridges will be made by both the OEM and licensees of the OEM, so a *chain of trust* is used

The Provisioning Process (Factory Setup)



ECDSA = Elliptic Curve Digital Signature Algorithm

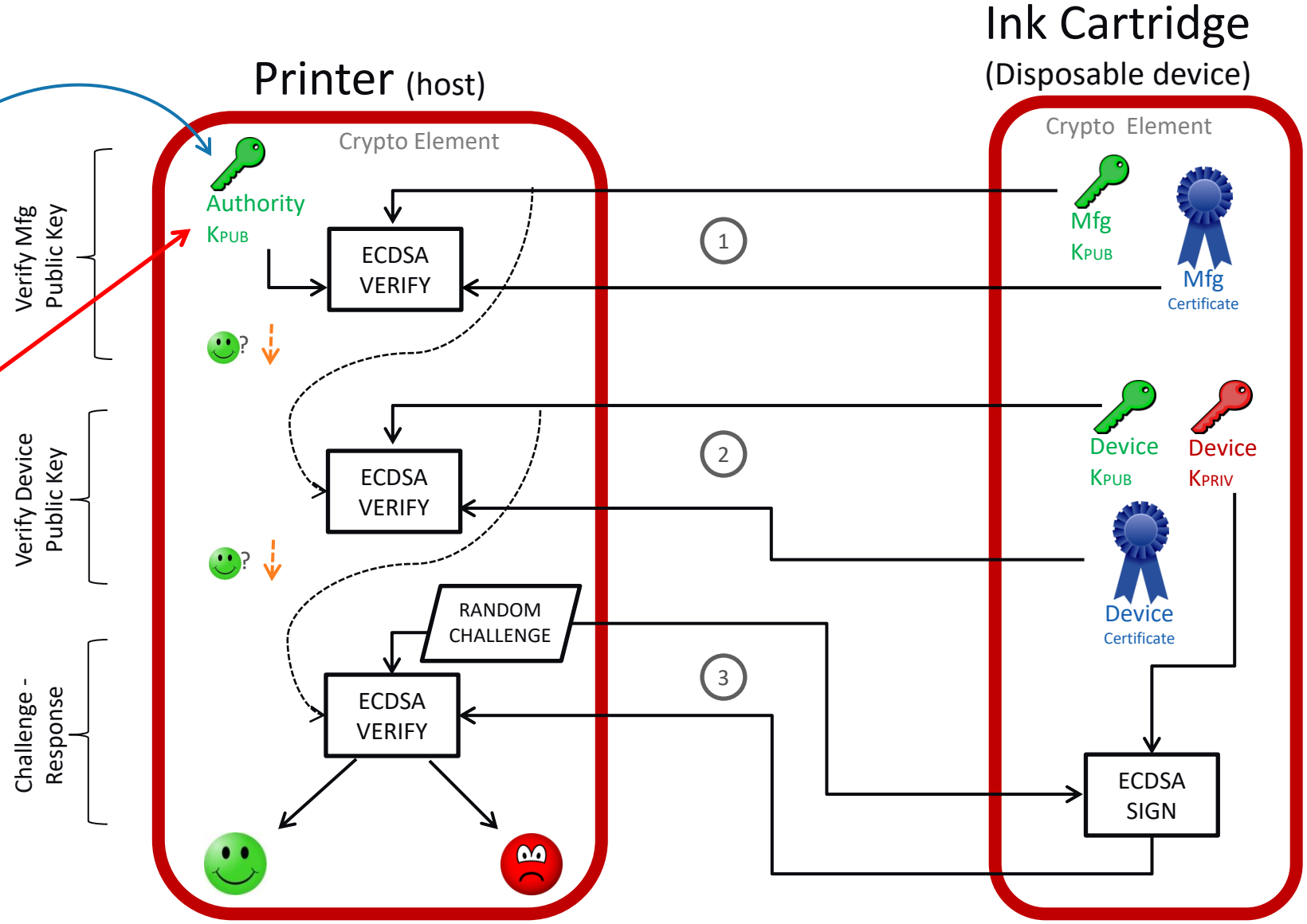
MICROCHIP

The Asymmetric Authentication Process

All hosts need knowledge of the authority public key

Why is the Root of Trust Public Key protected inside this crypto element?

Public keys are not secrets. Why does it need protection?

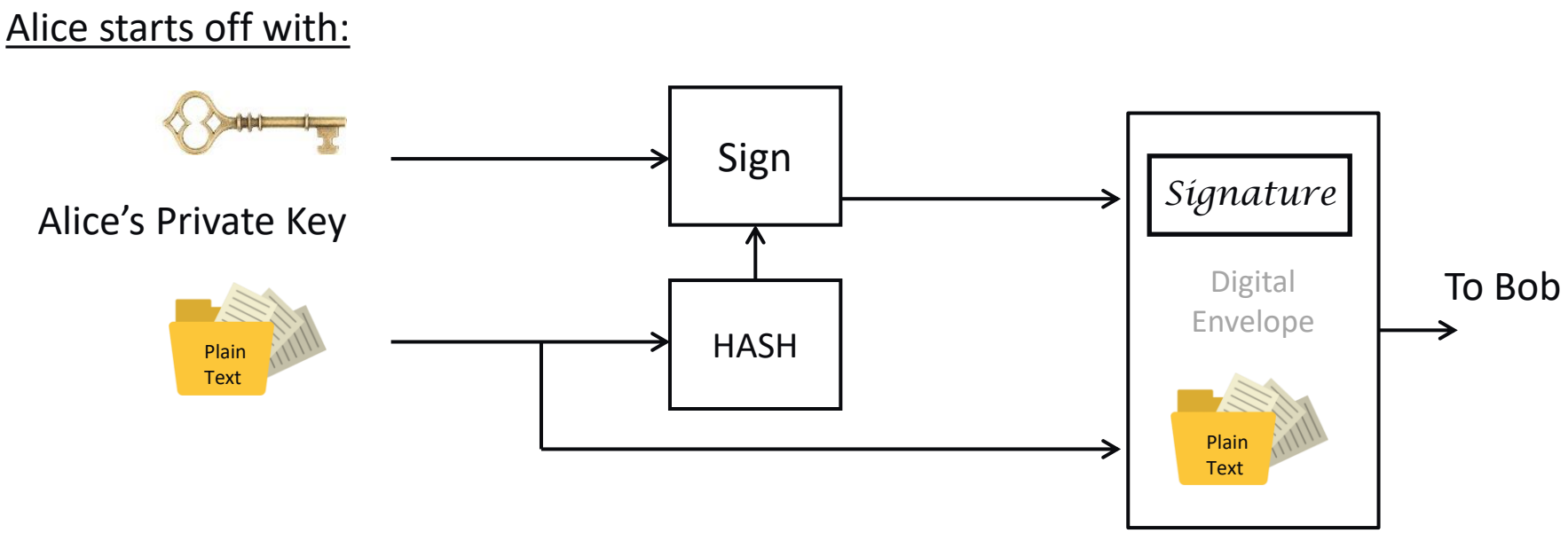


MICROCHIP

Creating a Signed Message

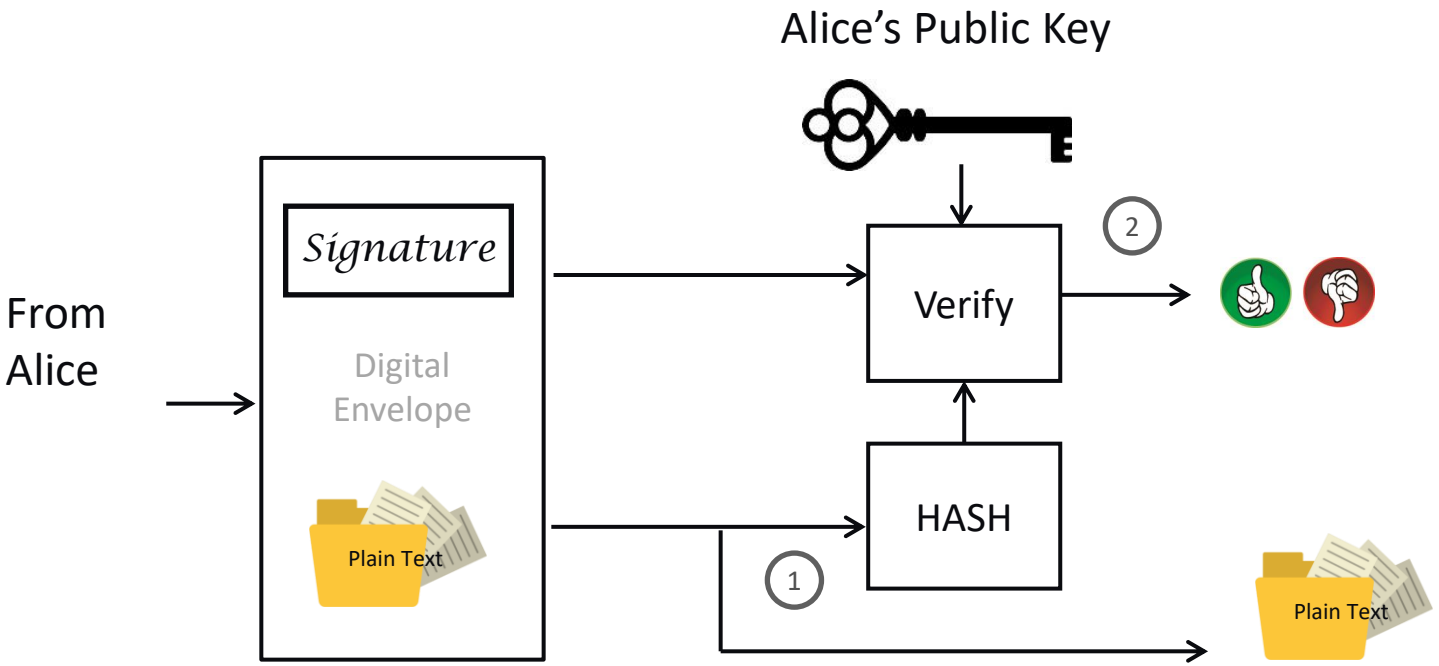
Bob wants assurance the file came from Alice and is not altered

NO ENCRYPTION



Receiving a Signed Message

Bob wants assurance the file came from Alice and is not altered



MICROCHIP

Encrypting Messages Using Diffie-Hellman

Alice wants to send Bob a secure message

Bob needs assurance the message came from Alice and is not altered

Alice calculates the ECDH shared secret

ECDH()() = **SS**

Alice's Private Key

Bob's Public Key

The same!

Bob calculates the ECDH shared secret

ECDH()() = **SS**

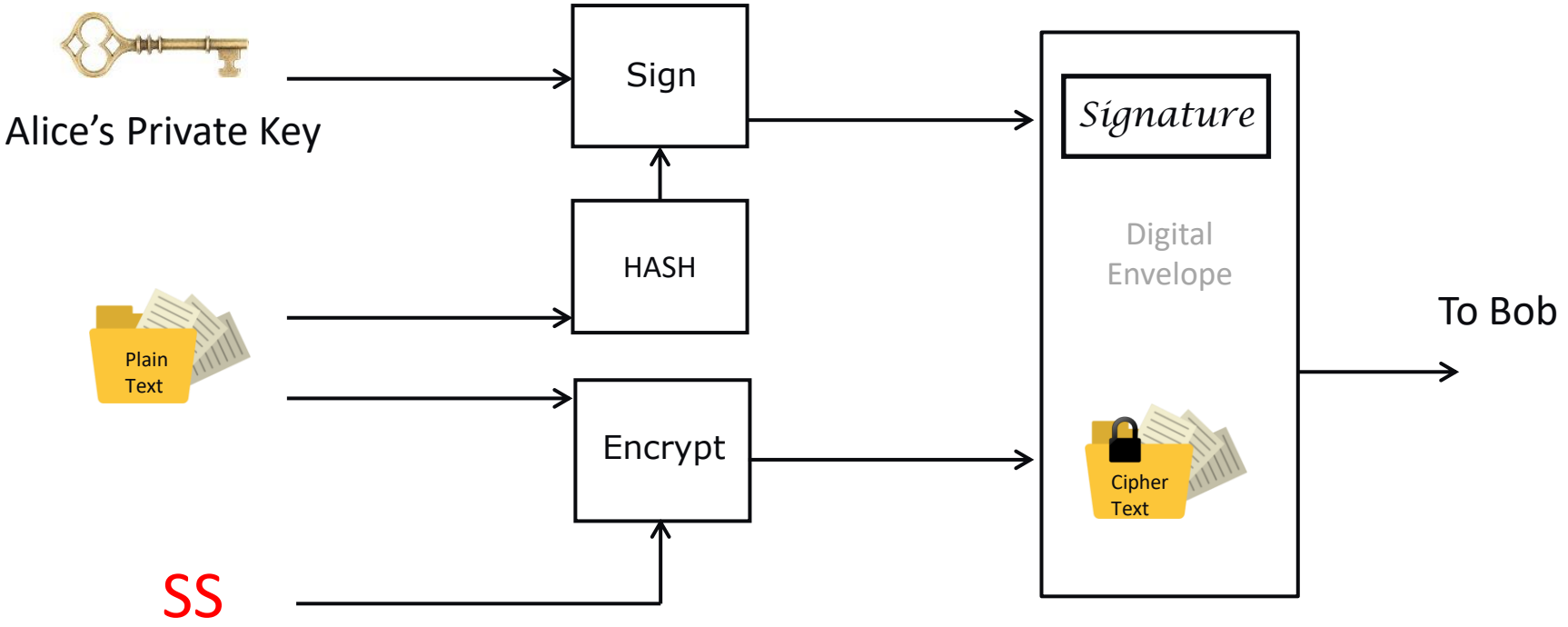
Bob's Private Key

Alice's Public Key

Encrypting Messages Using Diffie-Hellman

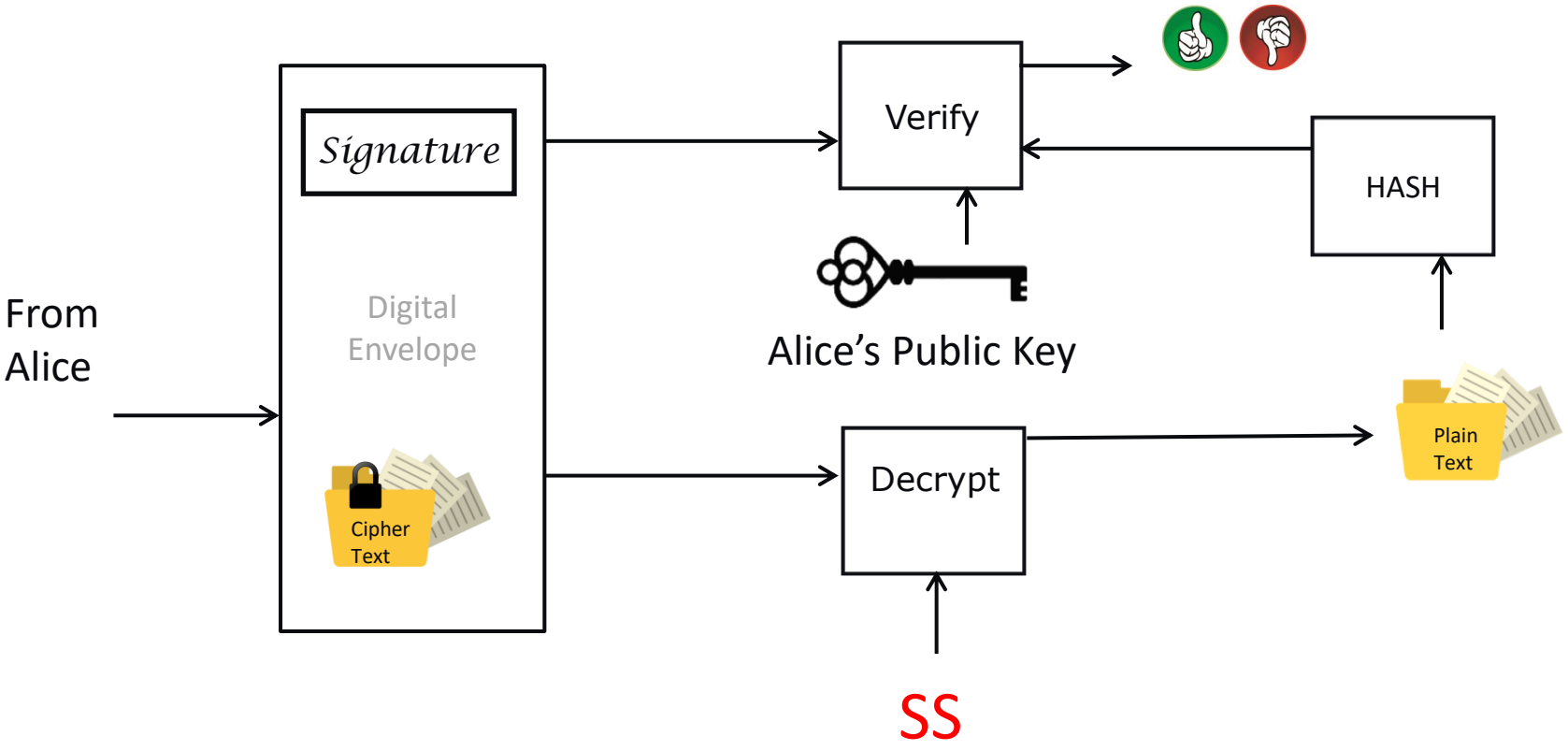
Alice wants to send Bob a secure message

Bob needs assurance the message came from Alice and is not altered



Receiving a Message Using ECDH

Bob wants to read the secure message Alice sent



With ECDH, “SS” is Always the Same

Shouldn’t We Have Different Keys for Every Session?

We could derive unique keys from our SS using a random number and a KDF (key derivation function), like we did in symmetric cryptography

OR

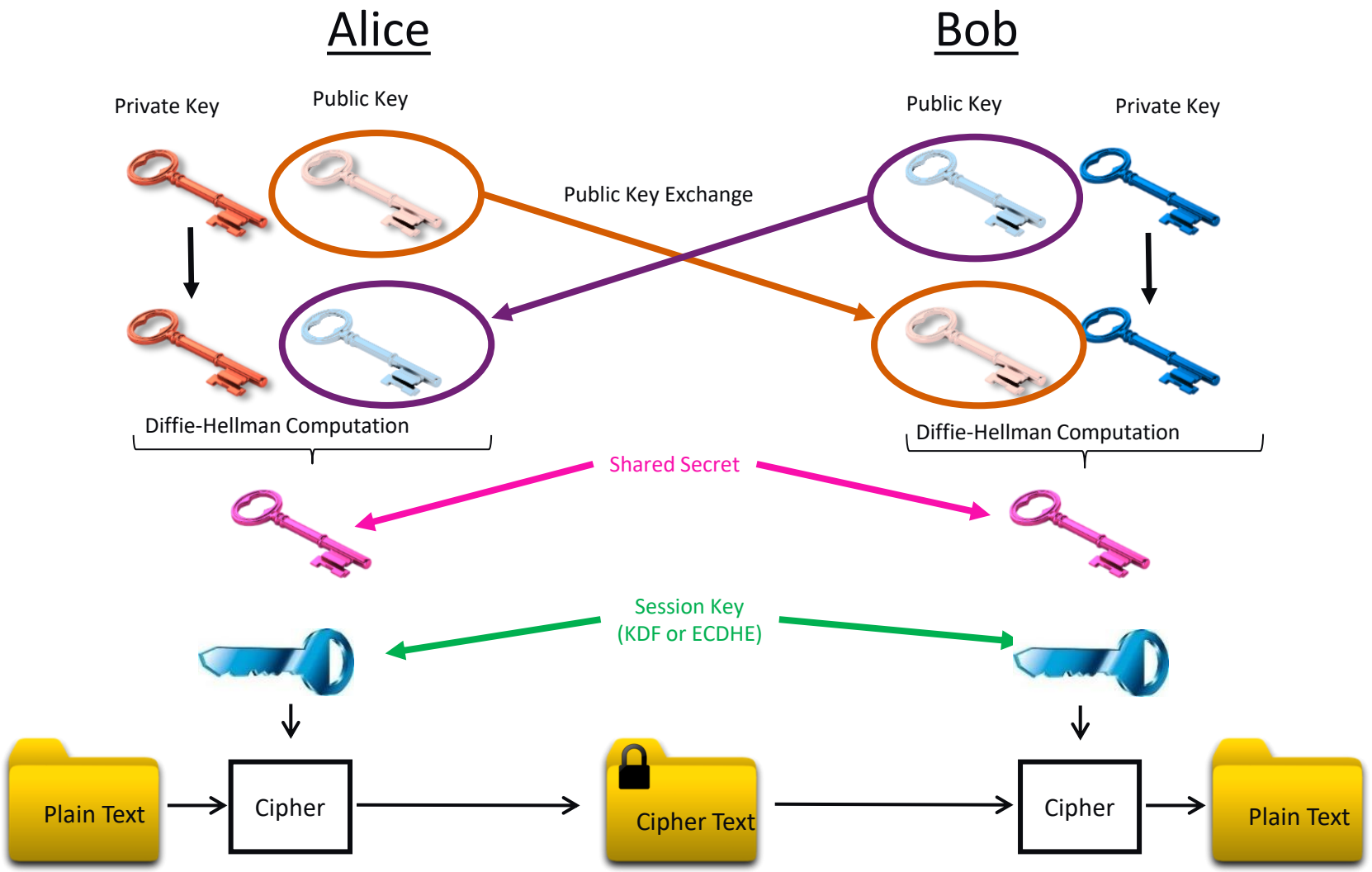
We can use Diffie-Hellman again to create ephemeral (temporary) session keys

- ECDH**E** – **E**phemeral (short lived) key generation
- ***After identity authentication***, each side spawns an ephemeral public/private key pair
  - These pairs are used with the ECDH algorithm to create a ***new shared secret***
  - The ***new shared secret*** is used as the session key / working key

After use, the *new shared secret*, and its ephemeral keys are destroyed

- Any information needed to reproduce the session key is destroyed forever
- The next time communication occurs, the above ECDHE process is repeated to create another ***session - unique new shared secret***

Derivative Keys Used in Communication



What Did We Just Cover?

- How asymmetric keys are related and used
- How authentication is performed using asymmetric cryptography
- How session keys are established
- How trust is established
- How session keys are spawned asymmetrically

Thank You