Presentation to M2MSec'14

ID-Based Design Patterns for M2M Secure Channels

Francisco Corella

fcorella@pomcor.com

Karen Lewison

kplewison@pomcor.com

Motivation

- The IoT is extremely diverse and gives rise to a broad range of requirements for secure channel protocols
- Requirements are often not met by traditional protocols such as TLS, IPsec or SSH
- Security is often sacrificed to meet constraints
  - BLE, ZigBee, Z-Wave are not secure
- There is a need for a broad variety of protocols to meet diverse requirements

Protocol design patterns

- Here we propose protocol design patterns rather than particular protocols
- Goal: reduce latency and bandwidth consumption by
  - Eliminating the roundtrips that most secure channel protocols require for key exchange
  - Eliminating certificate transmission
- Energy may be saved as a side effect of reduced bandwidth consumption, but that is not an explicit goal of these particular patterns

Previous work on reducing roundtrips and bandwidth consumption

- Caching parameters from initial connection can save roundtrips and transmission of certificates in subsequent connections, as in:
  - Abbreviated TLS handshake
  - TLS Fast-track
  - TLS False Start
  - TLS Snap Start
  - QUIC
- Proposed patterns:
  - Require no caching
  - Can eliminate roundtrips and certificate transmission for all connections

Pattern ingredients

- Identity-based encryption
  - A special case of identity-based cryptography, used for key transport
- Replay tolerance for first flow
  - So that initiator can send application data right away
- Optional forward secrecy from second flow onward
- For global deployments:
  - Hierarchical identity-based encryption with multiple roots
  - Reliance on DNS or DNSSEC to store ID chains

ID-based cryptography

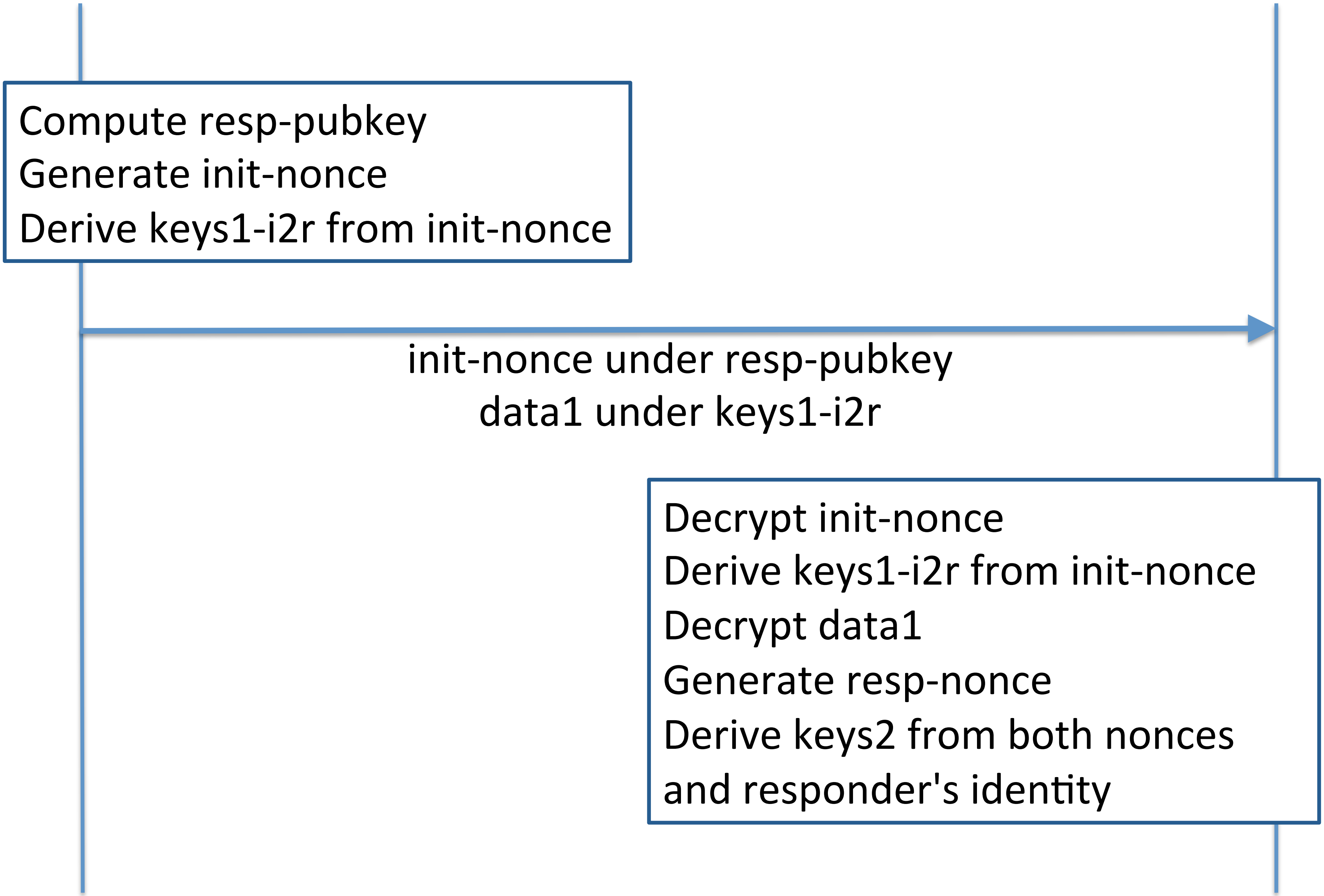
- Trusted party called “private key generator (PKG)” analogous to certificate authority (CA)
- Public key of entity is computed from ID of entity and public key of PKG
- Private key of entity is computed from ID of entity and private key of PKG
  - Must be computed by the PKG and given to the entity

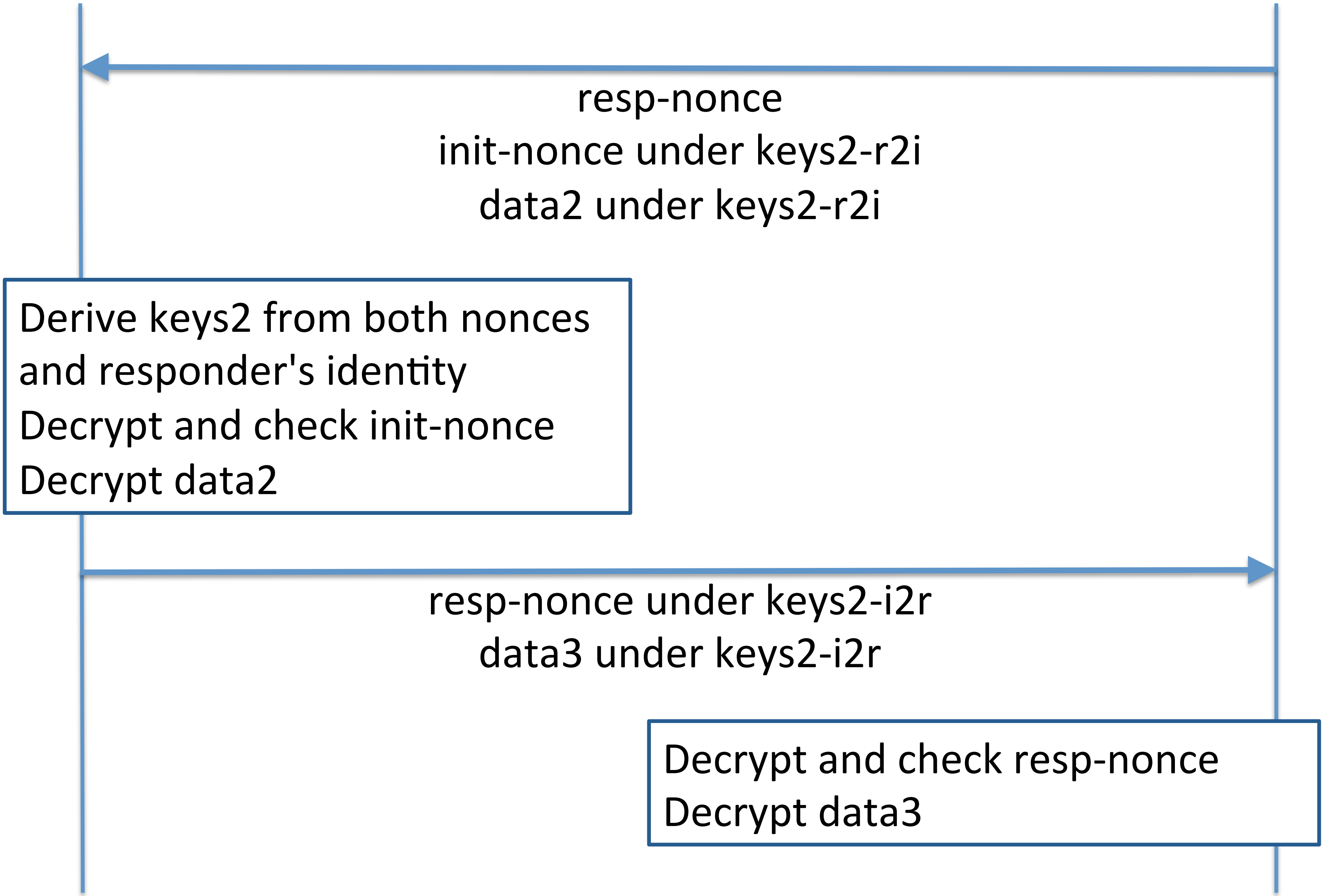
Expiration and revocation of ID-based credentials

- Key pair can be made to expire by adding an expiration date or time to the identity, e.g.
  - Base ID: "pomcor.com"
  - ID with expiration date: "pomcor.com-20141029"
- Key pair can be revoked by adding a revocation counter
  - ID with revocation counter: "pomcor.com-4"
  - *Latest ID must be retrieved from trusted repository*
- Short term expiration + emergency revocation counter
  - pomcor.com-20141029-4

Pattern #1

- Responder-only authentication
- No forward secrecy


Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce
init-nonce under resp-pubkey
data1 under keys1-i2r
Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Derive keys2 from both nonces
and responder's identity


resp-nonce
init-nonce under keys2-r2i
data2 under keys2-r2i
Derive keys2 from both nonces and responder's identity
Decrypt and check init-nonce
Decrypt data2
resp-nonce under keys2-i2r
data3 under keys2-i2r
Decrypt and check resp-nonce
Decrypt data3

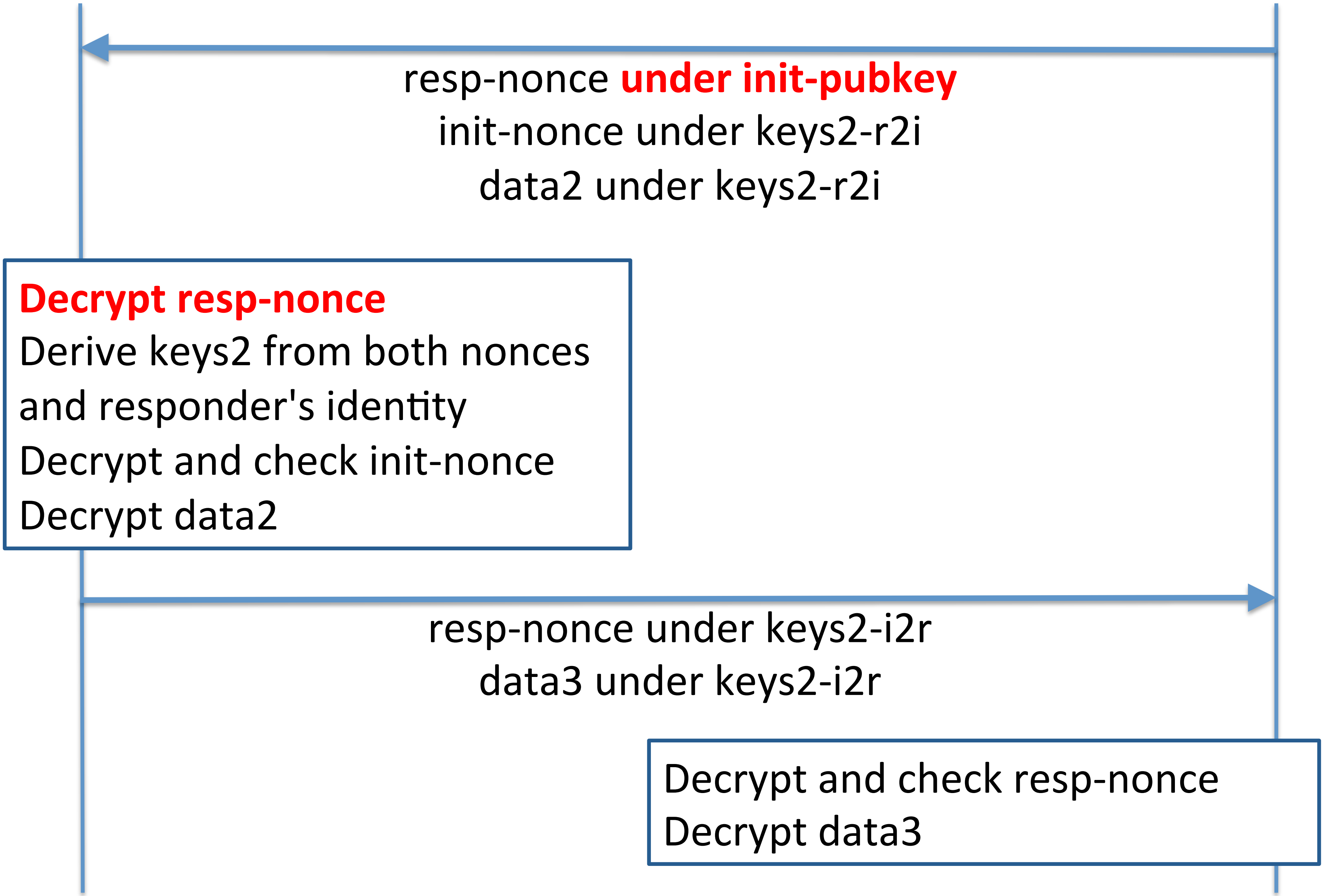
Pattern #2

- Mutual authentication
- No forward secrecy

Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce

init-nonce under resp-pubkey
data1 under keys1-i2r

Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Derive keys2 from both nonces
and responder's identity
Compute init-pubkey


resp-nonce **under init-pubkey**
init-nonce under keys2-r2i
data2 under keys2-r2i
Decrypt resp-nonce
Derive keys2 from both nonces
and responder's identity
Decrypt and check init-nonce
Decrypt data2
resp-nonce under keys2-i2r
data3 under keys2-i2r
Decrypt and check resp-nonce
Decrypt data3

Pattern #3

- Mutual authentication
- Forward secrecy from the second flow onward

Compute resp-pubkey
Generate init-nonce
Derive keys1-i2r from init-nonce
Generate init-edh-keypair

init-nonce under resp-pubkey
init-edh-pubkey
data1 under keys1-i2r

Decrypt init-nonce
Derive keys1-i2r from init-nonce
Decrypt data1
Generate resp-nonce
Generate resp-edh-keypair
Compute edh-secret
Derive keys2 from edh-secret
Compute init-pubkey

resp-nonce under init-pubkey
resp-edh-pubkey
init-nonce under keys2-r2i
data2 under keys2-r2i

Decrypt resp-nonce
Compute edh-secret
Derive keys2 from edh-secret
Decrypt and check init-nonce
Decrypt data2

resp-nonce under keys2-i2r
data3 under keys2-i2r

Decrypt and check resp-nonce
Decrypt data3

Other ID-based patterns

- Responder-only authentication with forward secrecy
- Patterns with initiator-only authentication…

For global deployments…

- A single PKG is not enough
- A hierarchy of PKGs may be needed
  - Analogous to a CA hierarchy
- A global PKG hierarchy may need to have multiple roots
  - Just like a global CA hierarchy
- The identity of a party is a chain of IDs of PKGs (the “ID chain”)
- The ID of a root PKG is a reference to cryptographic parameters built into the relying party
  - Analogous to a built-in root CA public key

DNS/DNSSEC support for global deployments

- The relying party needs to know what PKG chain is used by the authenticating party
- The ID chain of the responder could be retrieved from DNS/DNSSEC without an additional roundtrip
  - By contrast, a certificate chain is too large to be reliably retrieved from the DNS
- The revocation counter can also be retrieved from the DNS/DNSSEC
- Retrieving the ID chain from DNSSEC solves the "rogue PKG problem"
  - Just like DANE solves the "rogue CA problem"

Follow-up work

- Proofs of security for patterns
- Low-energy patterns?
- Design of protocols that use the patterns
- Proofs of security of protocols, relying on proofs of security of patterns as lemmas

Thank you for your attention!

For more information:
Web site: pomcor.com
Blog: pomcor.com/blog

Francisco Corella
fcorella@pomcor.com
Karen Lewison
kplewison@pomcor.com

Any questions?