Secure Messaging

...over unsecured transports

Agenda

1. Introduction and Background

2. Smoke Test:
   Lecture: Lab components walk-through
   Practical: Confirm installation of all lab components

3. CQ:
   Lecture: Traditional PKI vs DNS-bound PKI
   Practical: Establish DNS-based identity, self-test

4. 10-4:
   Lecture: Decoupled architecture and message security
   Practical: Use the pre-built docker app to pass signed and encrypted messages

5. Finding a Way Through, Around, or Over:
   Lecture: Deep dive on how the libraries under the lab environment work
   Practical: ~~Break into groups.~~ 😷

Introduction

About the Instructor

20+ (paid) years of experience:
Repair > Network engineering > Security > R&D, Protocol Development

Other Stuff:
Cellular security, various hardware hacking shenanigans

About You

Basic familiarity with DNS, DNSSEC, asymmetric cryptography and PKI

Working knowledge of Docker, Python, BASH

What You Will Learn

Identify a weak design pattern in message-oriented applications

Hands-on experience using DNS and PKI for messaging/object security

Optional: An implementer’s understanding of the above

What You Won't Leave With

Expert-level understanding of DNS/DNSSEC, PKI or cryptography

...unless you already had it when you walked in the door :-)

Disclaimer/Rules

The instructor (and associated orgs, etc...) accepts no responsibility for what you do with the tools. The tools and information are offered with the expectation that they will be used for research only, and not in violation of any laws or terms of service.

Know the laws of the jurisdiction you're in, and act accordingly.

Expect that the network is monitored.

Be respectful to one another.

Background and Motivation

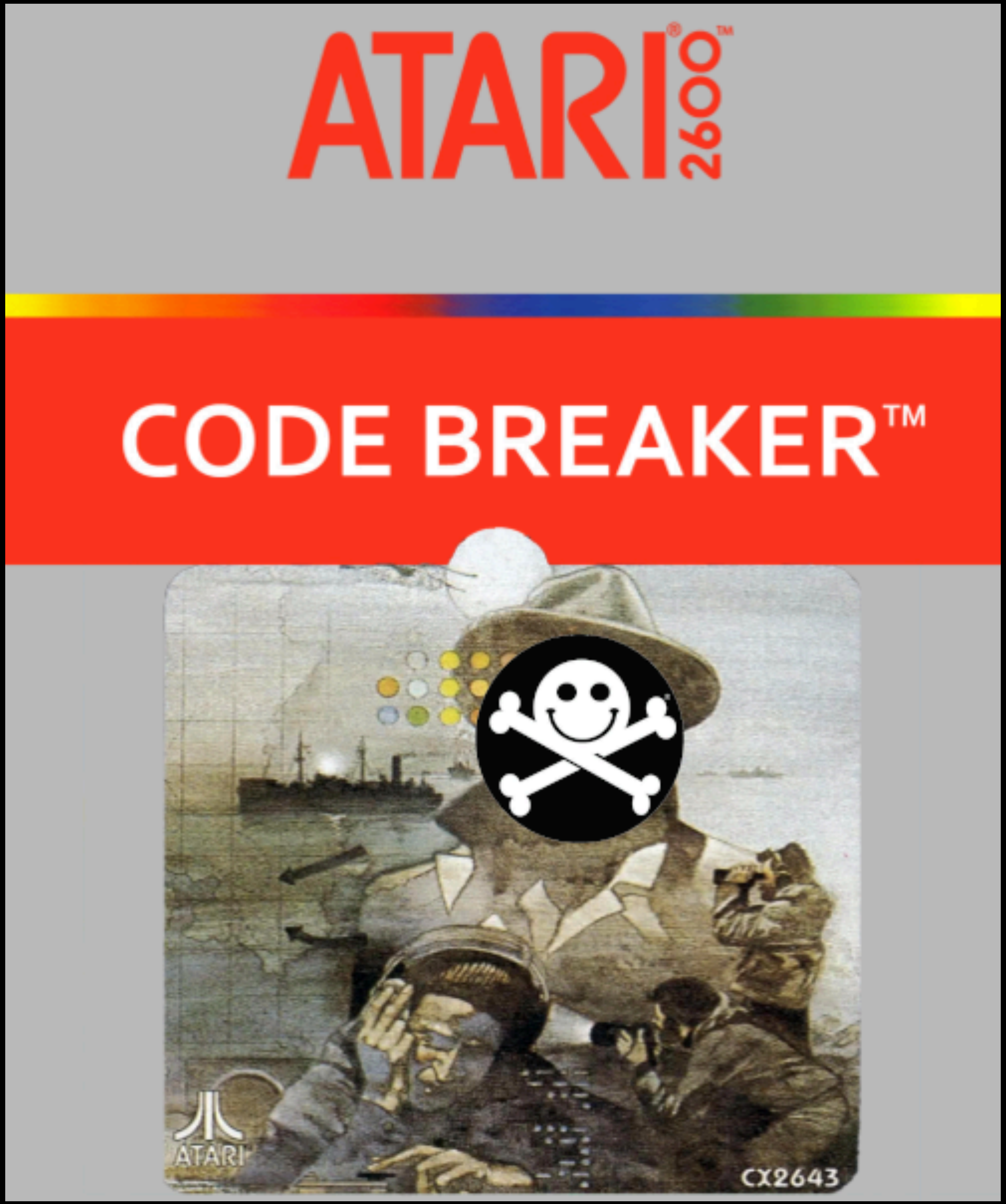
Absolute trust in messaging middleware for enforcing confidentiality/ integrity is a bad assumption.

Asymmetric cryptography can help to mitigate security issues in messaging middleware by providing E2E message security.

Standardized DNS record formats can make E2E security easier.

You Can't Stop The Signal

...And Now You Can't Distort It

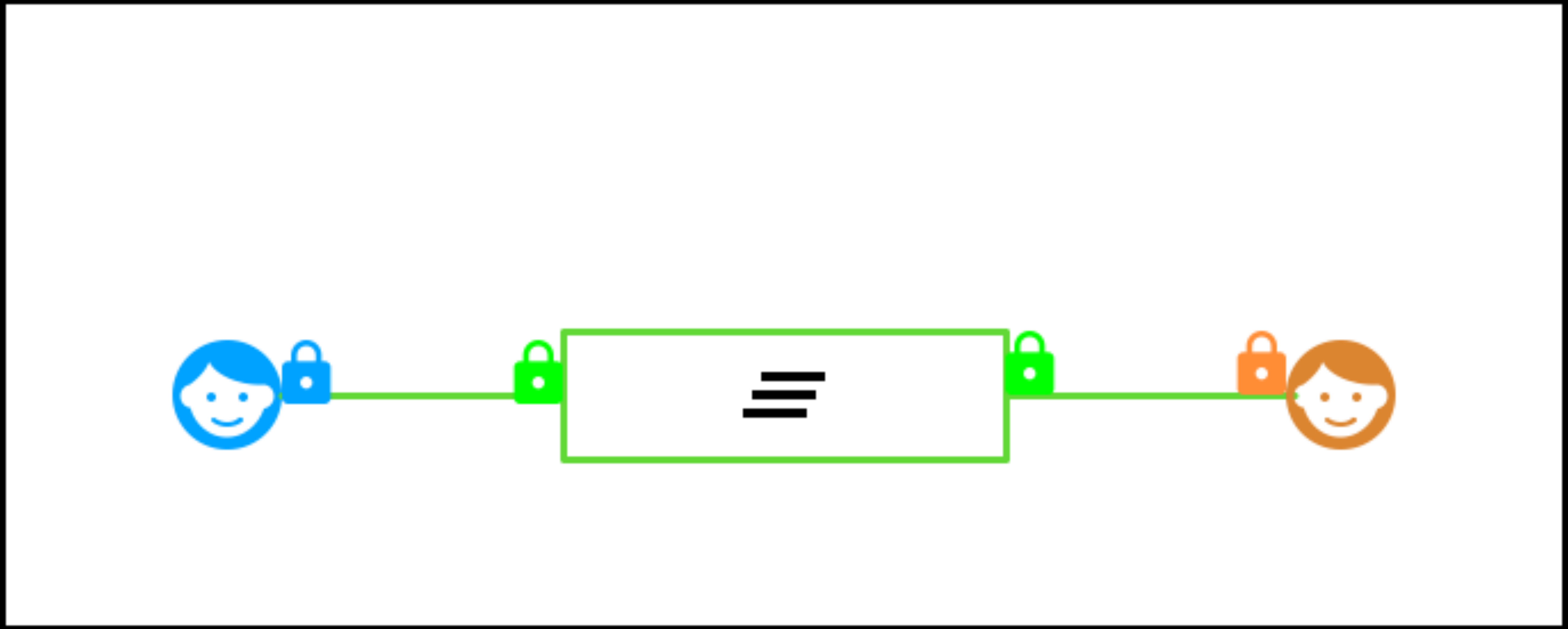


Communication Patterns

Client/Server:
- Both communicating peers authenticate one another.
- Peer identity can be established via certificate-based TLS mutual auth.

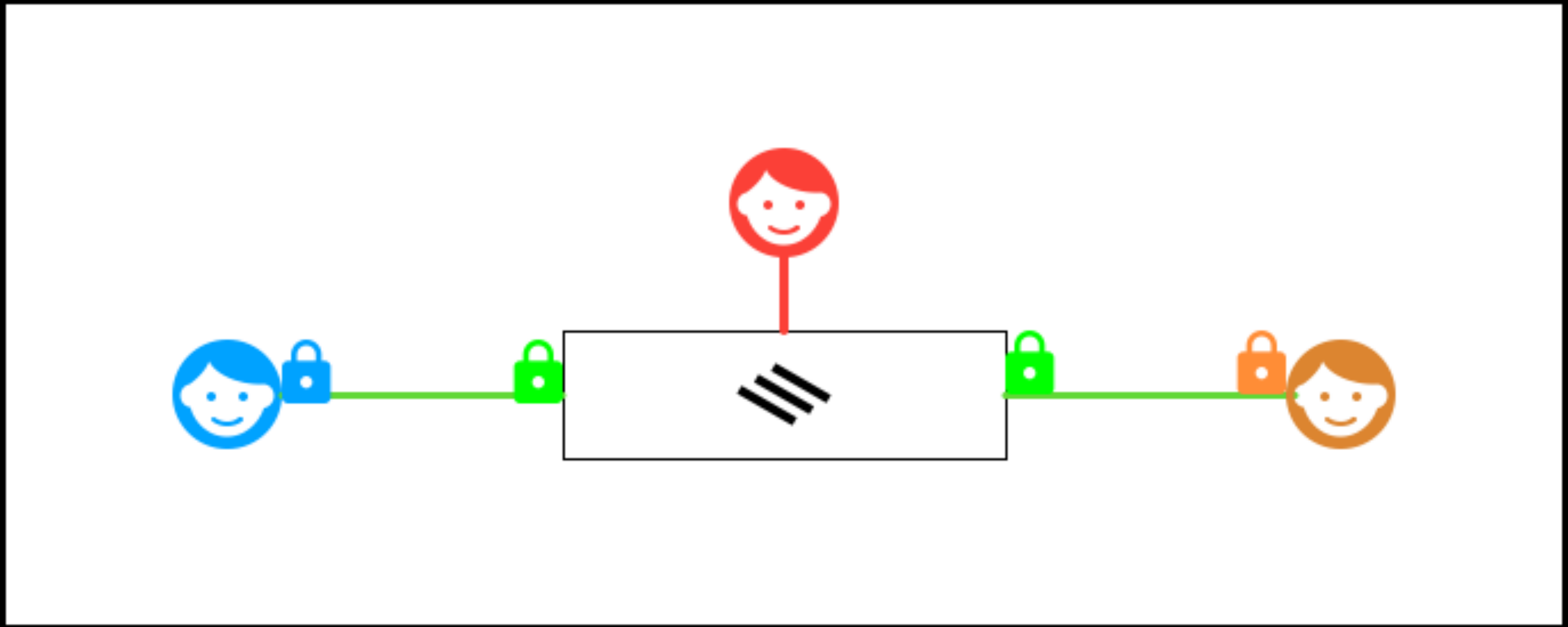
Publisher/Subscriber:
- Message-oriented interactions over store-and-forward systems.
- Oftentimes client/server patterns on either end of messaging middleware.
- Publisher and subscriber do not directly communicate.
- Trust in middleware for integrity/authenticity makes the middleware a high-value target.
- Peer identity ideally established via message security mechanisms.

Assumed Trust



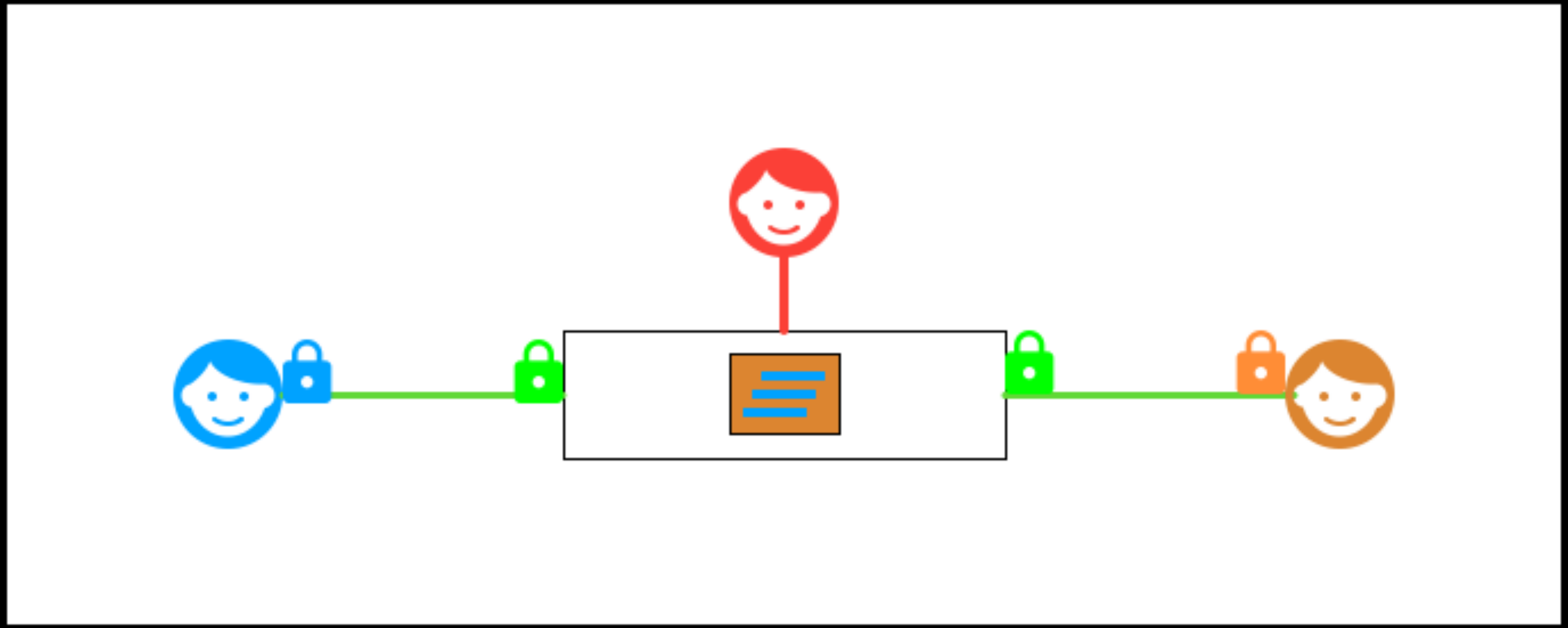
Availability
Integrity
Confidentiality

Assumed Trust



Availability
Integrity
Confidentiality

E2E



Integrity
Confidentiality

Availability

Integrity
Confidentiality

Digital Identity

Abstract

Digital Identity

Identifier

email address | @screen_name | commonName

Credential

password | biometric | certificate | PKI-enabled hardware token

Metadata

IRL name | phone number | friends | IP address

PKI and Asymmetric Cryptography

Asymmetric Cryptography: Alice and Bob exchange public keys
- Public keys enable encryption and signature verification
- How can we establish trust in public keys?

Public Key Infrastructure:
- Using public keys to establish trust in public keys.
- Certification Authority (CA): Org that validates identities and issues certificates
- Entity certificate: contains an identifier, public key, metadata, signature
- CA certificate: used to verify signatures on entity certificates

Entity names in PKI:
- Entity identifiers are only guaranteed unique within the scope of a CA.
- Trusting multiple CAs makes impersonation across CAs possible.
- This is due to the lack of a single, unified namespace for PKI.

Alice and Bob

Alice and Bob both trust the CA

Alice and Bob exchange public keys enclosed in certificates signed by the CA

A message is signed by Alice's private key

The signed message is encrypted using Bob's public key

Bob uses their private key to decrypt the message, then uses the public key in Alice's certificate to verify the message signature

DNS and DNSSEC

DNS is the universal namespace of the internet

- Delegated hierarchy
- Universal read-only access
- Write access only for domain owners

DNSSEC is a PKI, bound to DNS

- DNS root zone KSK is the trust anchor for DNSSEC

DANE

DNS Authentication of Named Entities:

- Convey a public key (certificate, public key, or hash) via DNS
- Present usage context in the same DNS record:
  - Certificate could be an entity certificate or CA certificate
  - Can signal that the client must auth server using PKIX + DANE

Binds a public key (or certificate) to a TLS service running on a specific port.

The DANE protocol uses DNS to establish:

- Name constraints across CAs
- Public key lookup by DNS name

DANE Illustrated

```
dev123._device.example. 20 IN TLSA 3 0 0 308204C5308202ADA00302010202142CEEE5D9
                                   | | |   |
                  Certificate Usage | |   |
                           Selector |   |
                        Matching Type   |
                    Certificate Association
```

DANE Illustrated

```
dev123._device.example. 20 IN TLSA 3 0 0 308204C5308202ADA00302010202142CEEE5D9…
                                   | | |    |
                   Certificate Usage | |    |
                              Selector |    |
                           Matching Type    |
                     Certificate Association
```

Certificate usage: How should the certificate be used?

0: The record refers to a CA certificate for the entity. This CA certificate must also pass PKIX validation.
1: The record refers to an entity certificate. The presented certificate must match and pass PKIX validation.
2: The record refers to a trust anchor certificate, which the entity must be able to prove a chain of trust with.
3: The record refers to the exact certificate which must be presented by the entity.

DANE Illustrated

```
dev123._device.example. 20 IN TLSA 3 0 0 308204C5308202ADA00302010202142CEEE5D9…
                                    | | |    |
                   Certificate Usage | |    |
                            Selector |    |
                       Matching Type    |
                  Certificate Association
```

Selector: What part of the entity certificate should be matched against the DNS record?

0: The entire presented certificate should be matched
1: Only the subjectPublicKeyInfo portion of the certificate should be matched

DANE Illustrated

```
dev123._device.example. 20 IN TLSA 3 0 0 308204C5308202ADA00302010202142CEEE5D9…
                                   | | |   |
                   Certificate Usage | |   |
                             Selector |    |
                          Matching Type    |
                     Certificate Association
```

Matching Type: How should the presented certificate be matched against this record?

0: The DER encoding of the presented certificate must match against the Certificate Association field
1: The SHA-256 hash of the presented certificate must match against the Certificate Association field
2: The SHA-512 hash of the presented certificate must match against the Certificate Association field

DANE Illustrated

```
dev123._device.example. 20 IN TLSA 3 0 0 308204C5308202ADA00302010202142CEEE5D9…
                                   | | |    |
                   Certificate Usage | |    |
                              Selector |    |
                           Matching Type    |
                     Certificate Association
```

Certificate Association: What should be matched against the presented certificate?

This field contains either a DER-encoded certificate or the hash of a DER-encoded certificate, which must match the presented certificate, within the constraints provided by the prior 3 fields.

DANE Illustrated

```
dev123._device.example. 20 IN TLSA 3 0 0 308204C5308202ADA00302010202142CEEE5D9…
                                   | | |    |
                   Certificate Usage | |    |
                              Selector |    |
                           Matching Type    |
                     Certificate Association
```

Taken together:

The **entire** **entity certificate** must **exactly match** the **Certificate Association field**.

Smoke Test

Lab Environment

Local workstation:

Docker

Git

Public services:

DNS/DNSSEC (you provide)

MQTT (I provide)

If you don’t already have this pre-configured, now is a GREAT TIME

Our First Basic Application

Since we trust DNS(SEC):

Alice and Bob use DNS to lookup public keys for encryption and authentication

Alice uses MQTT to publish messages for Bob

Pre-Flight Check

```
git --version
    2.29.2

docker -v
    20.10.7

docker-compose -v
    1.29.2
```

https://dnssec-analyzer.verisignlabs.com/${YOUR_DOMAIN}

    If you can't do DNSSEC, raise your hand!

Pre-Flight Check (alt)

If you can't configure DNSSEC for your zone:

You must have an HTTPS server with a browser-recognized certificate here:

`https://device.${YOUR_DOMAIN}`

(now is a good time to grab that Letsencrypt cert…)

Static content will be hosted here:

`https://device.${YOUR_DOMAIN}/`

(go ahead and place a file here, test with your browser)

Download the source code

```
git clone https://github.com/ux00c6/dane-messaging-workshop

cd ./dane-messaging-workshop

cp ./.env.example ./.env

vim ./.env
```

Variables in the .env file

IDENTITY_NAME

DNS name of your messaging application

Follow this pattern: `${whatever}._device.${YOUR_DOMAIN}`

TRUSTED_DOMAINS

Just put ${YOUR_DOMAIN} here for now.

MQTT_HOST

MQTT_PORT

1883

Build the Application

```
docker-compose up --build -d && docker-compose logs -f
```

CQ

Comparing PKI and DANE

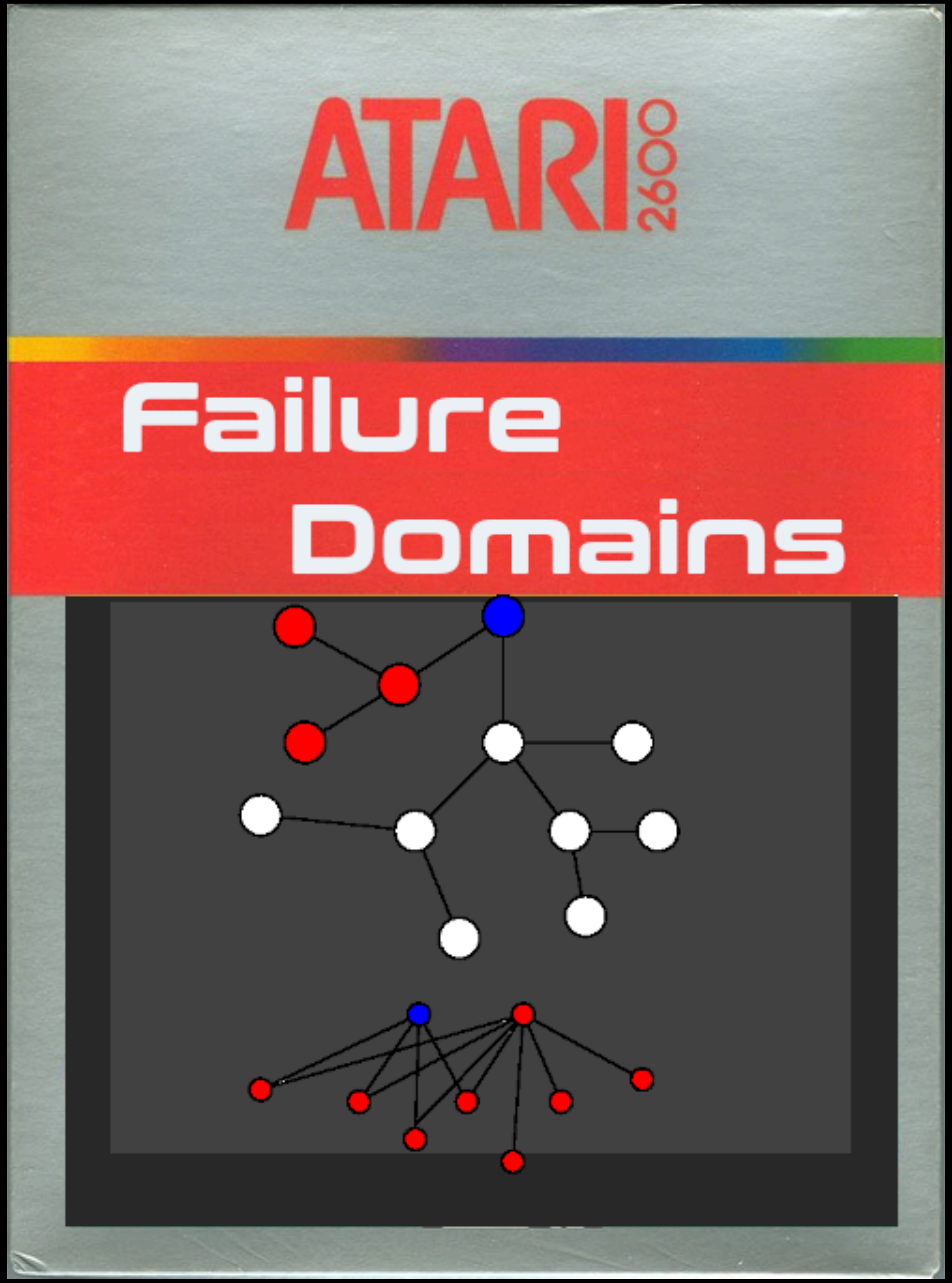
DNS: Trust anchor associated with root zone

Web PKI: CA bundle contains many trust anchors

DNS zone compromised:
Anything below that point in the DNS namespace can’t be trusted.

PKI trust anchor compromised:
Anything trusting the compromised trust anchor has a big problem.

Bogus SSL certificates have been a problem in the past, which led to the creation of Certificate Transparency logs (detection, not prevention).



PKI and DNS

Authentication with DNSSEC: **abc123._device.example**

Retrieve the certificate via DNS, require DNSSEC before accepting

Oftentimes recursive resolver is trusted for DNSSEC verification

A better approach is for the stub resolver to verify the response

PKI and DNS (alt)

Authentication without DNSSEC: **abc123._device.example**

Retrieve the certificate via DNS, extract the dNSName attribute.
Match the cert dNSName to the DNS name used to retrieve it.
Extract the authorityKeyID from the certificate.
Construct the URL* to locate trust chain:
    https://device.example/authorityKeyID.pem
Use the trust chain to validate the certificate.

*HTTPS server must present a browser-recognized certificate!

Bootstrapping DNS-Based Identity

```
docker exec -it identity_manager

./generate_selfsigned_identity.sh

./generate_tlsa.sh
```

Bootstrapping DNS-Based Identity (alt)

```
docker exec -it identity_manager

./generate_selfsigned_identity.sh

./generate_tlsa.sh
```

Host certificate PEM at
  https://device.example/a-k-i.pem
  Where AKI is the authorityKeyID from the cert. (look in the terminal history)

Loopback test

Run the command:

```
./cache_public_identity.sh ${YOUR_IDENTITY}
```

10-4

Examples of Decoupled Architecture

Email

Microservices

Kafka, MQTT

Distributed IoT (LoRaWAN, Edge)

Blogs and forums

Email (PKI)

S/MIME (RFC 5741): Use PKI for email verification/encryption

DANE + S/MIME: RFC 8162

Use DANE TLSA records for S/MIME certificate discovery

Email (OpenPGP)

OpenPGP: Public key signing/encryption, web of trust

DANE + OpenPGP: RFC 7929

Use DNS for OpenPGP public key discovery

Microservices, Decoupled

Message brokers: pub/sub interaction patterns

The message broker is often where verification ends.

Using DANE for public key discovery enables payload verification, even across organizations, without distributing CA certificates

IoT, edge

Edge computing: decoupled architecture which can enable information to be summarized or processed close to where it originates.

Challenges similar to microservices, but add bandwidth and power constraints

How the Application Works

Create a message, including the sender’s DNS name

Attach a signature over the message and sender name

Get the recipient’s public key from DNS

Encrypt the message using the recipient’s public key

Publish the encrypted message on an MQTT topic containing the recipient’s DNS name.

Recipient listens on MQTT topic containing own IDENTITY_NAME

Recipient decrypts message

Recipient compares sender DNS name against allowed domains

Recipient uses DNS to get sender public key and verify message signature

MAKE A FRIEND

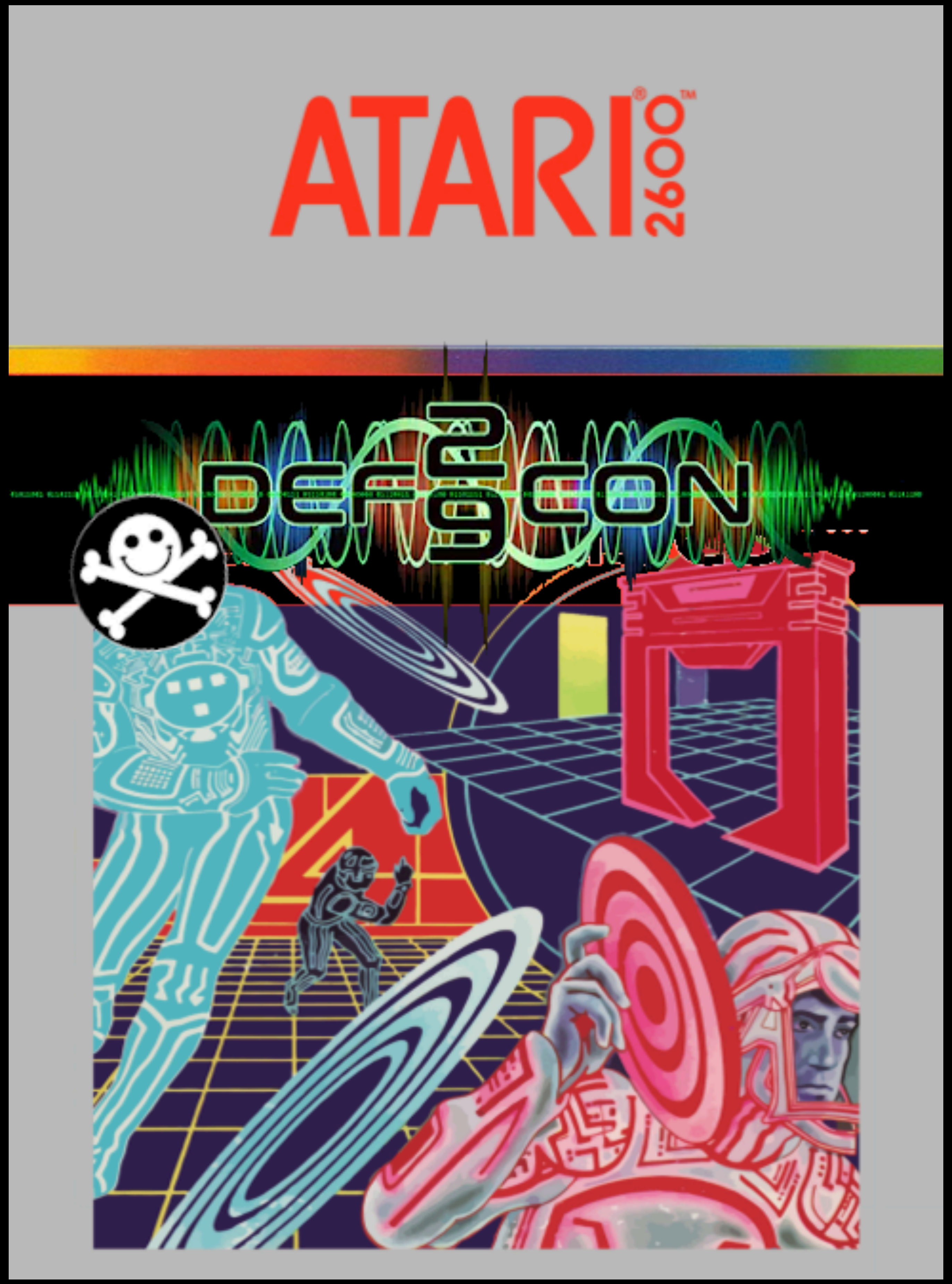
Pick another attendee and exchange IDENTITY_NAME

Edit your .env file and add your new friend’s DNS domain to TRUSTED_DOMAINS (comma-separated)

Send a message:

```
docker exec -it message_sender “Hello World” ${RECIPIENT_IDENTITY_NAME}
```

Find a Way: Through, Around, or Over



Build Something New

Underlying Tools

Python is not necessarily required

OpenSSL used for generating certs and PKI

OpenSSL can be used to generate signatures and encrypt messages

Generate > Sign > Encrypt > Transmit

Receive > Decrypt > Verify > Process

REQUIREMENTS

Ability to publish data
- Format constraints?

Signaling mechanism for recipient
- Does publishing medium provide signaling, or do we need out-of-band?

THANKS

Conthrax Font: https://www.fontspring.com/sku/TDC1156919

Atari 2600 cover art: https://archive.org/details/RetroboxReadyAtari2600CoverArt

Emoji: https://openmoji.org/

Google Material Design icons: https://fonts.google.com/icons