Module 0

Pre-Reading

corda

Introduction

Corda builds upon key insights and tools from the fields of distributed ledger technology and cryptography. This pre-reading pack will provide you with the necessary grounding in these areas to fully understand Corda's architecture and internal workings.

Contents:

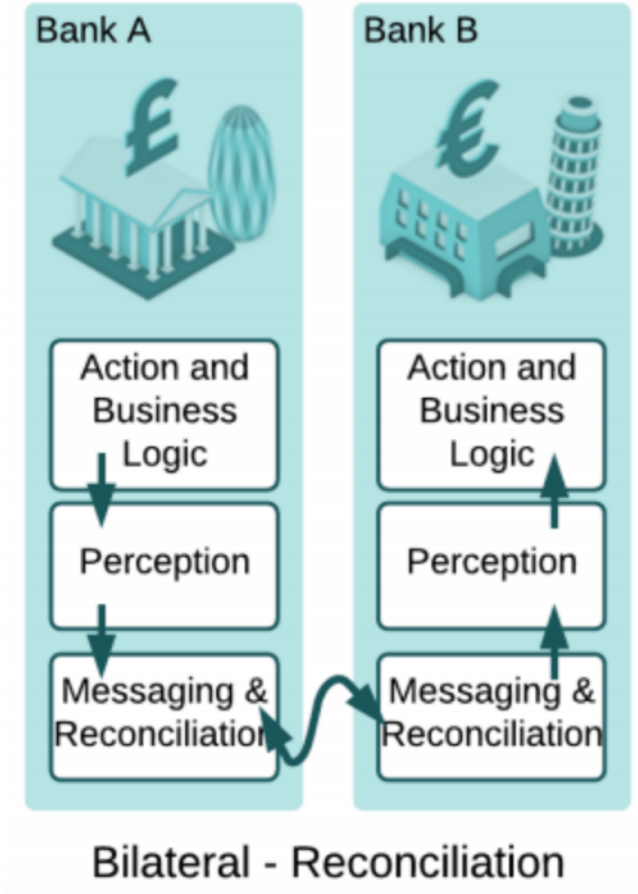
Distributed Ledger Technology

The Reconciliation Problem

Today, each organization maintains its own **ledger**, which records that firm's view of its agreements and positions with respect to its customer set and its counterparts.

This duplication leads to inconsistencies, driving the need for costly matching, reconciliation and fixing of errors. Inevitably, differences will remain between firms' views of the same transaction, which is also a source of (potentially systemic) risk.

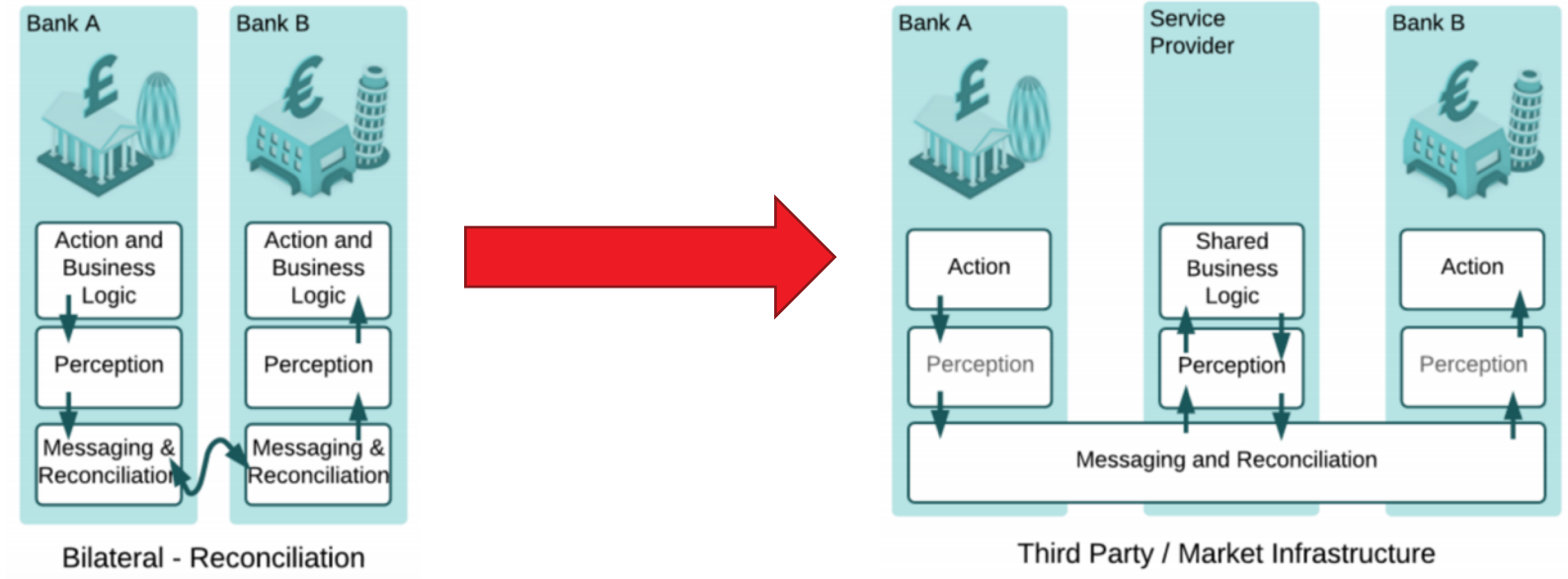
As a result, firms must continuously engage in the process of **reconciliation**. This is manual, error-prone and time-consuming endeavour.



Centralized Databases

An alternative would be for firms to hold their data in a single centralized database:

- Reconciliation would be eliminated
- Firms would reduce risk by improving the accuracy of their view of the world
- Technology costs would fall



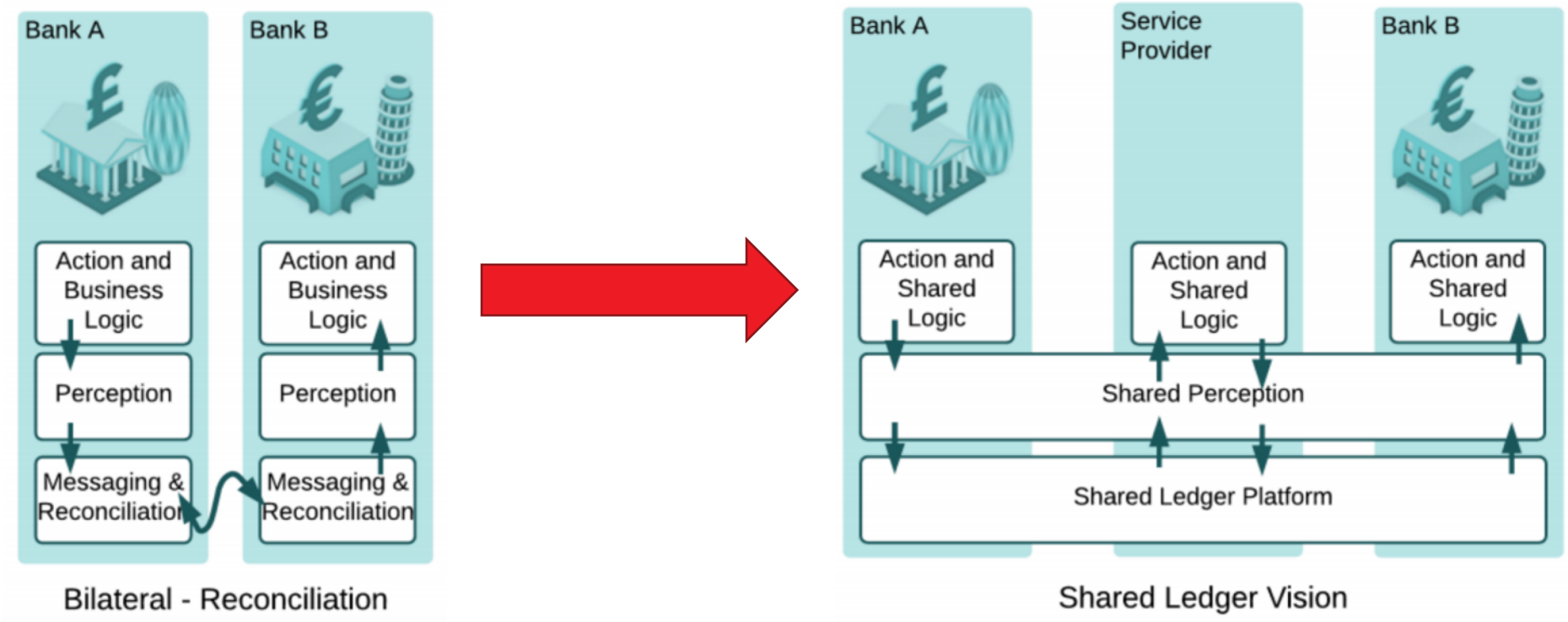
Issues with Centralized Databases

In practice, however, there are major barriers to the adoption of centralized databases:

- Who would run this database?
- Who would own it?
- In which jurisdiction(s) would it be hosted?
- What would stop those jurisdictions abusing the mountain of sensitive information it would have?
- What if it were hacked?
- Can you actually scale a relational database to fit the entire financial system?
- What happens if The Financial System™ needs to go down for maintenance?
- How would you guard changes to the database schemas?
- How would you manage access control?

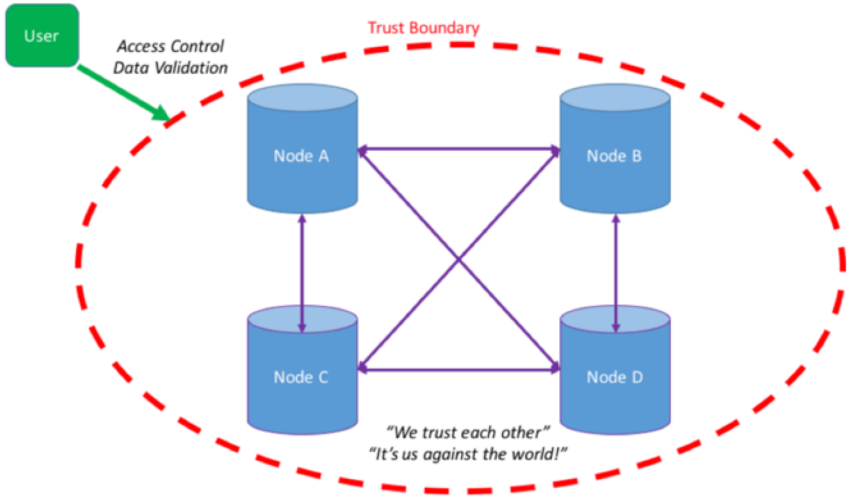
Distributed Ledgers

Distributed ledgers allow for a shared ledger **without** centralized control:

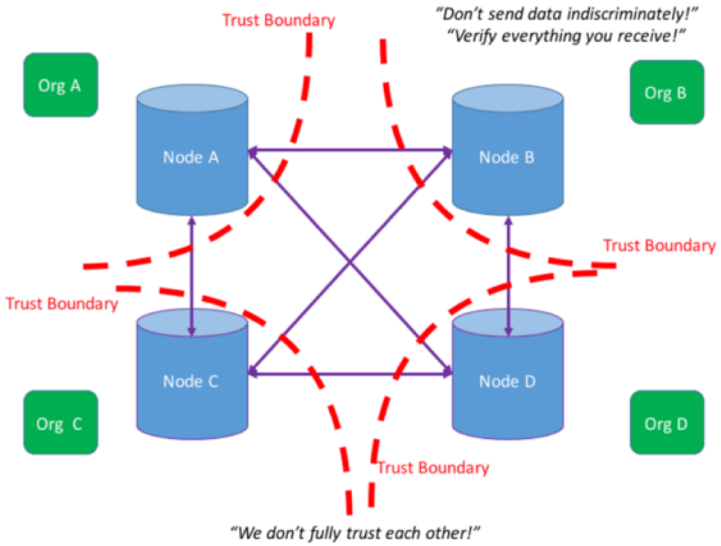


Trust Boundaries

With a distributed database, the trust boundary encompasses all the database nodes – they **MUST** all trust each other. A distributed ledger alters the trust boundaries, allowing nodes to significantly reduce the amount of trust they place in others.



Distributed Database

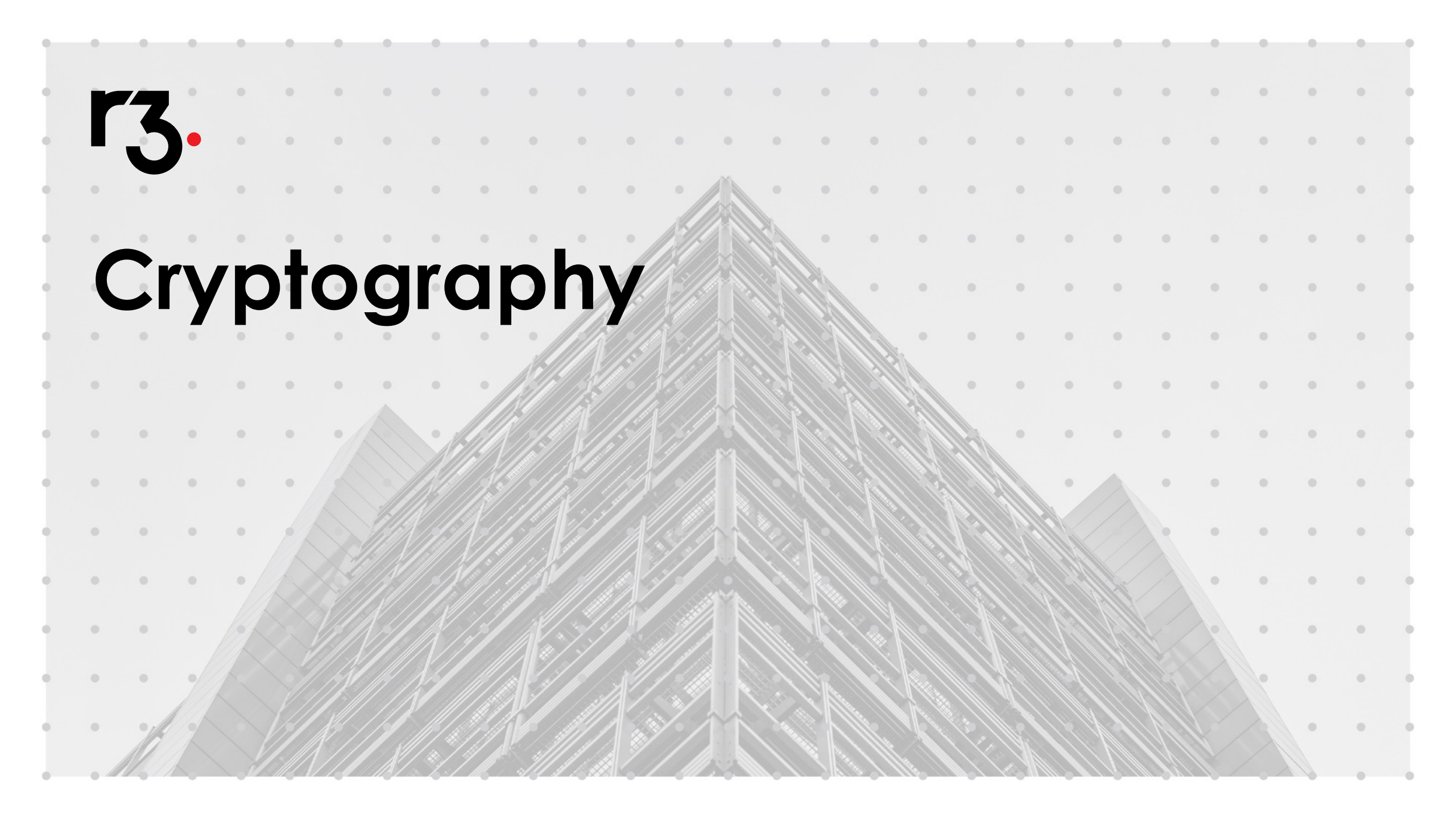


Distributed Ledger

Characteristics of Distributed Ledgers

All distributed ledgers share several key characteristics:

- **They offer a single view of shared facts**
  - Either a single ledger used by everyone
  - Or separate ledgers, but with a mechanism to ensure consistency
- **Rules govern the evolution of the ledger**
  - Transactions
  - Contracts
- **Consensus mechanisms resolve competing ledger updates**
  - Proof-of-work
  - "Pluggable" consensus



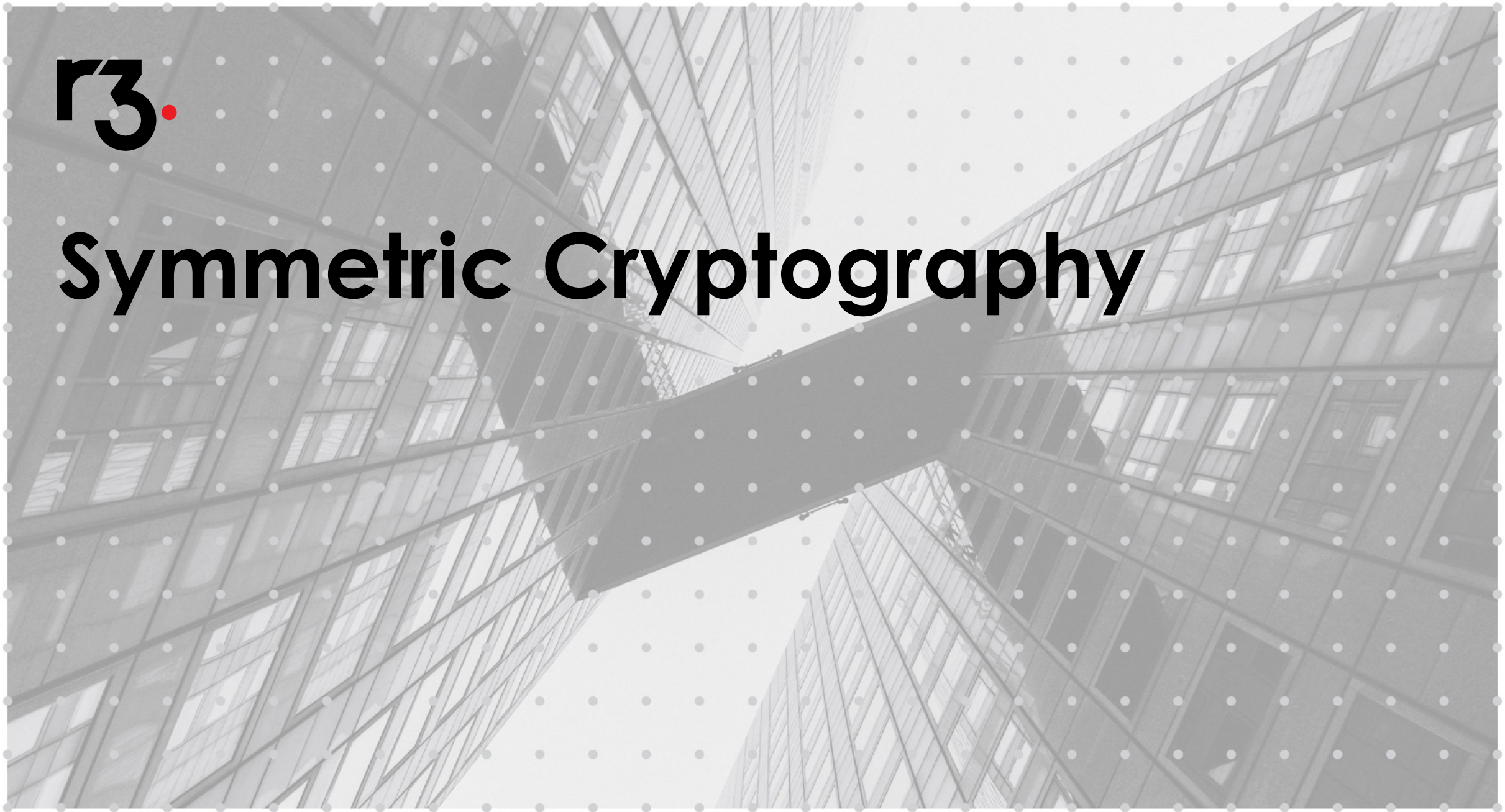
Cryptography

Cryptography Introduction

Cryptography concerns sending secret messages that eavesdroppers can't understand or tamper with.

Terminology:

- **Plain text** is readable
- **Encryption** is the process of turning plain text into **cypher text** (which looks like gibberish!) using an **encryption key**
- **Decryption** is the process of turning **cypher text** back into **plain text** using a **decryption key**



Symmetric Cryptography

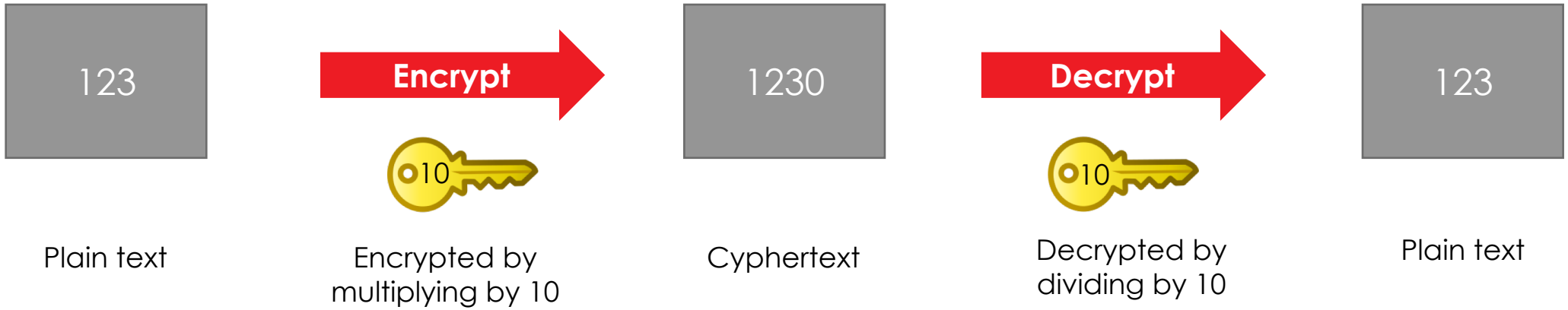
What is Symmetric Cryptography?

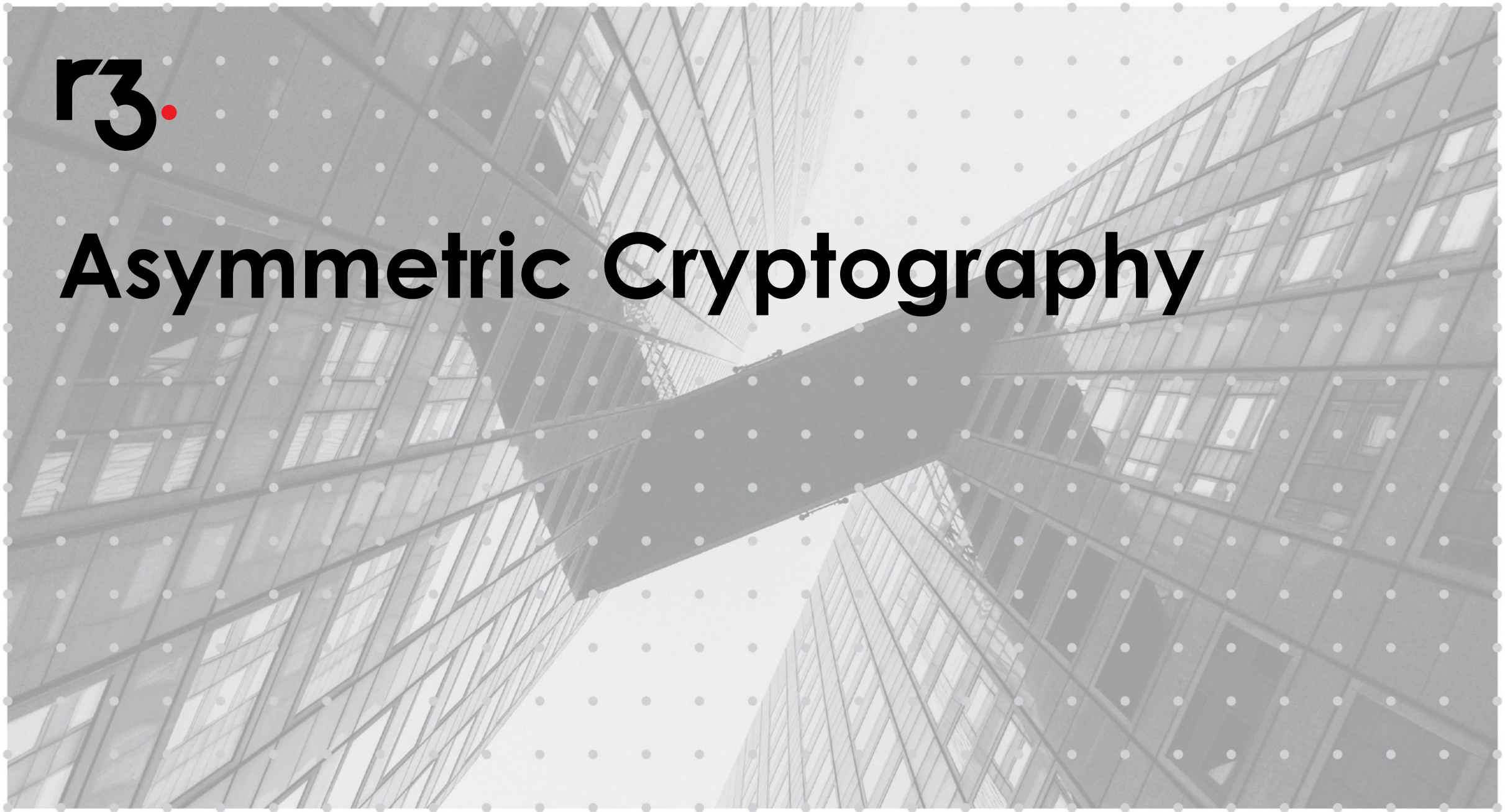
With symmetric cryptography, the encryption and decryption keys are the **same**.

A really **bad** encryption technique might be:

- **Encrypt** by multiplying the input by the number 10
- **Decrypt** by dividing the cypher text by the number 10

This is symmetric cryptography because the same key (10) is used for encrypting and decrypting.





Asymmetric Cryptography

What is Asymmetric Cryptography?

With asymmetric cryptography, the encryption and decryption key are **different**, but mathematically linked.

Asymmetric cryptography works as follows:

- A pair of related keys (known as a **key-pair**) is generated
- The **first** key is used to **encrypt** the plain text
- The **second** key is used to **decrypt** the cypher text

Public and Private Keys

Public key: The first key – the one used for encryption - can be put on a website or a company directory. It is called a public key because anyone can access it. You need to give this to people so they can encrypt messages meant only for you.

Private key: The second key – the one used for decryption - is held securely by only the message recipient, who uses it to decrypt the cyphertext. If you lose this, you can't decrypt the message. If someone copies it, they can decrypt and read messages meant for you.

It's the security of the private key that is important. You can calculate the public key from a given private key, but not the other way around.

Example Public/Private Keypair

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAwKLLxLHKnKVjluSYFxolfWmiI6d8kpNchTWJPjh+H5PhH65/
bzz9X1w0Pnr5G+HFleUjtshmuvWH80sG5b5WiZRpSBaBo6vcGpehItJk1SUumQbW
p9/LPIxIu9QOYlKqVbRIRy4gLfpUdq6eMBLRA8vYk4wpVWv0v+PdAv32WtRvW/P7
eTAbyFQPvRhzG6uuPfGnk51VzOM83H58S4fB6pQNEbzgBC/d3RxH4mdDz3oj9aRa
HXJJ6PEhH5mCouGcSSAJkCbXnr8gvFZVV9Nwj+zDBe8ks7RMBE0yS3RN8S18NJcb
/vhMVq9S3wphUKApgxsprzLR6ZcK+5TZ/EpQZQIDAQABAoIBACK/ux46LCBhvTtE
9SKFtumOmo7v8YrHRlNJOv6cVjQACwYaBibcy5/eFBaJPFKJQYkukmiwcceb7vA2
QoJC+X1L5kn4tV3+7G9nlsufRE0mrecw0pvpxNGhRKWNNSMLYmqfYMJmUau5km7u
RN7LJiUXKnRDPlmIZdgPlJirkzsC8pqvHcdWyikQdpzfnrve8XhLMHMKAl0JFsBe
zMf4LcXY9IZUV+b+T/RGK+CddZJf0Shu8mWMz39XLap1/Y+m29mF2I/Vc1+d1pf7
6DekMgjqe5aRczX2Fjd4s9NkgSWQT9EPwjfwFxjN0X8IAXuVEbVXrngTZuVDHiSG
PDS1UWkCgYEA9tTMyokiZmwvHwLKTYIHUj9lJv+80dZsilfaL6Gb7Vde4fw0segB
Ho6wOnQec+tY5L/4Oatz6I2M96rHux3ih8fO35fEcgFH1vBVNLA4WCXTXCSAx9lQ
o6ERq6lLJ0gCWuHTu47tafnmtXruqFsrRaPzZ05FeSnw8KoKjAsPBGMCgYEAx8qi
e2tisRNUBjkyImgETaHaZjKNM6UkHIzsbW6cO93bVlJy9CIyqvtPKXphzCB5AFHH
Fx+BzPoPpOUgmfdSw5hzYFSApNLA0pIh85RpNdcvAI/QdmsOjTL3X2SysGHZosmn
5BqWRhV6IdrqeAwjfAHyGnl8dlB5YJ3t5kiCfpcCgYEArNXBOKjhv5pNPIe1pKRC
ACOH7tDwJHbitdRiYSvmhUDCnbb+Jvzvm7gKSkP49u565mJNrAQUO2iyptGc3bdF
YhJVF1esJLAk36qV3zjtmzuRTln5GvdecdVdDDiU+UKA86ZhnlmsgKKOG9Giv5xn
JnJxA4tZNkhaAArxK7Kqp0UCgYAlHnCxnXHolD64KUg3PDwQcFGzSxEQ102uQEBs
1HASb1Wg5BCXawMe4TID8sjYs+/HL2pd0Wy8OHlPmYJFGL43uMs2ynxcIwvQE3Zb
WudxKVmCMTX5ykoZ0a5D+uF4A7X7mJuotYfV3YejYpjNVPQCxhnE8W0+ThU+7hiV
AEO+YQKBgQDNfeS0LZ0zIVmSuOpTlgV8FeWoNn/5m1efKzAzCBWoh9D+SN1VxQ8F
wYRK0ejl1gj+zXrVqe0kYWEoBpjNhdoC2t6z/2hKNNf9nqD/nBdqM8qX7sDRH6l3
955BtY9jw4nhDDNNLqNmLgGiPrSA80kUz+OQACpTwUENKOmfFIy3sw==
-----END RSA PRIVATE KEY-----
```

Public key:

```
AAAAB3NzaC1yc2EAAAADAQABAAABAQDAosvEscqcp
WOW5JgXGiV9aaIjp3ySk1yFNYk+OH4fk+Efrn9vPP
1fXDQ+evkb4cWV5SO2yGa69YfzSwblvlaJlGlIFoG
jq9wal6Ei0mTVJS6ZBtan38s8jEi71A5iUqpVtEhH
LiAt+lR2rp4wEtEDy9iTjClVa/S/490C/fZa1G9b8
/t5MBvIVA+9GHMbq6498aeTnVXM4zzcfnxLh8HqlA
0RvOAEL93dHEfiZ0PPeiP1pFodckno8SEfmYKi4Zx
JIAmQJteevyC8VlVX03CP7MMF7ySztEwETTJLdE3x
LXw0lxv++ExWr1LfCmFQoCmDGymvMtHplwr7lNn8S
lBl
```

Digital Signatures

Public/private keypairs can also be used to create **digital signatures**.

You **sign** the hash of a message using your private key, producing a digital signature. You send the the digital signature alongside the message.

Someone else can take the signature, along with the message and your public key, and validate the **authenticity** and **integrity** of the message:

- **Authenticity**: The digital signature proves that "the private key matching this public key was used".
- **Integrity**: The digital signature proves that the message was not tampered in-flight. If the message is tampered after being signed, the digital signature is no longer valid.

The digital signature is only valid for the message it signs, so you can't copy a digital signature and reuse it with a different message.

Non-Repudiation

Digital signatures also provide **non-repudiation**.

Example:

- Alice and Bob wish to enter into a contract
- Alice provides a digital signature of the contract to state that she agrees to its terms
- Alice later decides to renege on the contract, and states that she never signed it
- Bob checks that decrypting Alice's digital signature with Alice's public key outputs the contract's original plain text of the contract
- Thus the digital signature could only come from the holder of Alice's private key – in other words, Alice!

Alice can only claim that she never agreed to the contract if her private key was stolen, or if the keypair which provided the original signature never belonged to her to begin with.

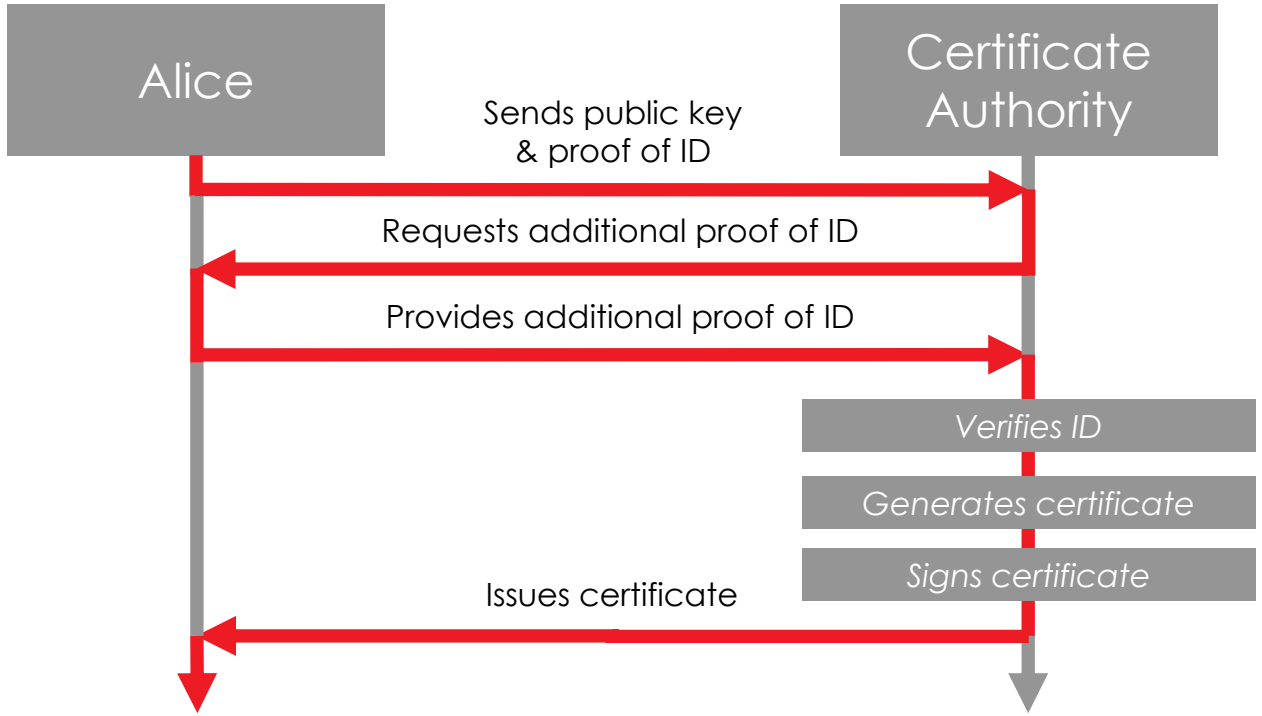
To avoid these situations, we need a way to properly link keypair holders to identities. In practice, this linkage is provided by a public key infrastructure – the topic of our next section.

Public Key Infrastructures

Public Key Infrastructures

A **public key infrastructure (PKI)** allows public keys to be bound to real-world identities using **digital certificates**.

New digital certificates are generated by trusted **certificate authorities (CA)**:



Example Digital Certificate

Data:

Version: *1 (0x0);* ***Serial Number:*** *7829 (0x1e95)*

Signature Algorithm: *md5WithRSAEncryption*

Issuer: *C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc, OU=Certification Services Division, CN=Thawte Server CA/emailAddress=server-certs@thawte.com*

Validity: *Not Before: Jul 9 16:04:02 1998 GMT; Not After : Jul 9 16:04:02 1999 GMT*

Subject: *C=US, ST=Maryland, L=Pasadena, O=Brent Baccala, OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org*

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit): 00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb: 33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1: 66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66: 70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17: 16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b: c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77: 8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3: d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8: e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

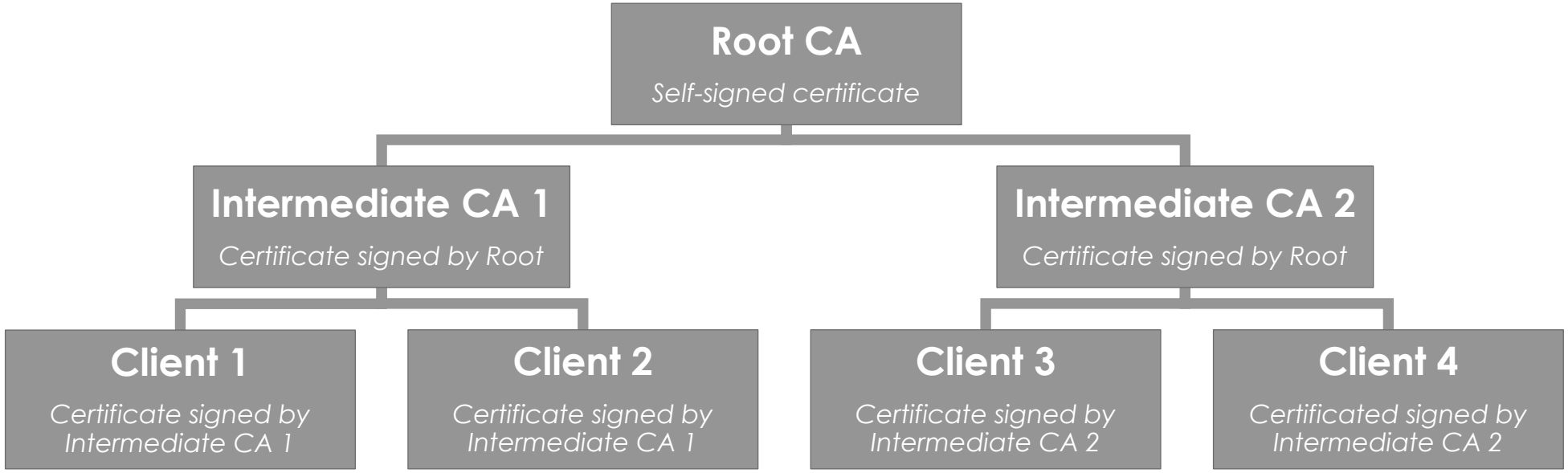
Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d: 92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92: ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67: d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72: 0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1: 5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7: 8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22: 68:9f

The Chain of Trust

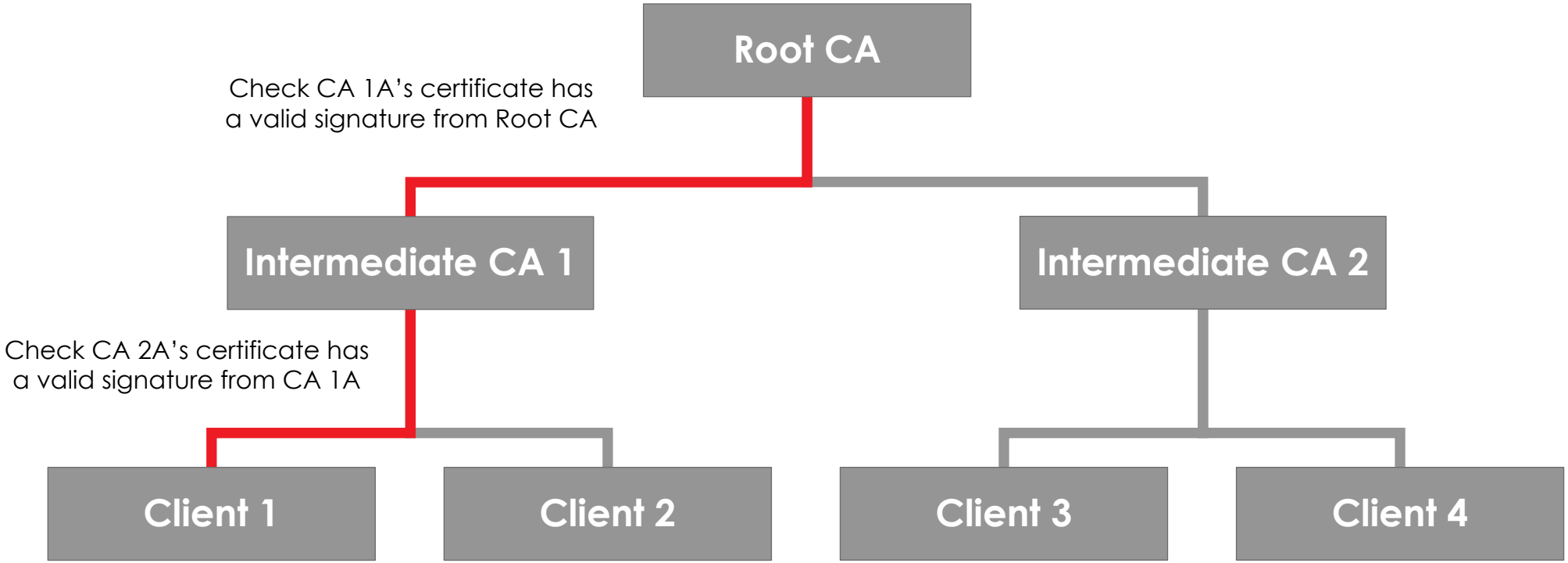
CAs are organized into layers, with the CAs on layer *n* signing the certificates of the CAs on level *n + 1*. This is known as the **chain of trust**.

The CA on level 0 is the **root certificate authority**, on which the authority of all the other CAs ultimately depends.



Walking the Chain

When you receive a certificate signed by a CA on level *n* > *0*, you have to walk the chain of trust back to the root CA to ensure its trustworthiness:



Hash Functions

What are Cryptographic Hash Functions?

A hash function is a mathematical operation that transforms input data into an output digest.

The input data can be of any size/length, and the output hash is usually a fixed length. There are many types of hash functions (MD5, SHA-1, SHA-256…)

What does a hash digest look like? Here is the string "The cat sat on the mat" put through two different hash functions:

Property 1 – Hash Functions Are Deterministic

Property 1: Hash functions are deterministic.

That is, they always give the same output for a given input. The MD-5 hash of "The cat sat on the mat" is and will always be 001203ab59e23d266d71eb29873aa7.

The digest is sometimes called the *hash* or the *fingerprint* of the data (but not the *signature*)

Property 2 – Hash Functions Are One-Way

Property 2: If you change the input just a little bit, the output changes unrecognizably.

This makes it hard to "back calculate" the input data if you know know the hash. This "one way" or "trapdoor" property is important in cryptography.

Property 3 – Hash Functions Are "Collision Resistant"

Property 3: It is hard to find two inputs that hash to the same output.

We call these identical outputs for different inputs "collisions".

As there is an infinite number of possible inputs, and only a fixed set of digests (because digests have a fixed length), there will always be different inputs that hash down to the same digest. However, good hash functions minimize the number of such collisions.

Property 4 – Hashes Are “Hiding”

Property 4: If you know the hash digest, you should not be able to determine or infer anything about the input (its length, whether its an even or odd number, the file-type…).

A hash function with this property is called “hiding”.

Other Properties of Hash Functions

Other properties:

- A hash function should be computationally efficient (i.e. take a small number of calculation steps to perform)
- The hash digests should be well distributed (i.e. they shouldn't display patterns, such as starting more often with a "5")

You can hash more than just text files - you can hash any data including entire hard drives.

Note that the hash is **not** "*encrypted*" or "*compressed*", since it is not possible to retrieve the original data from the hash.