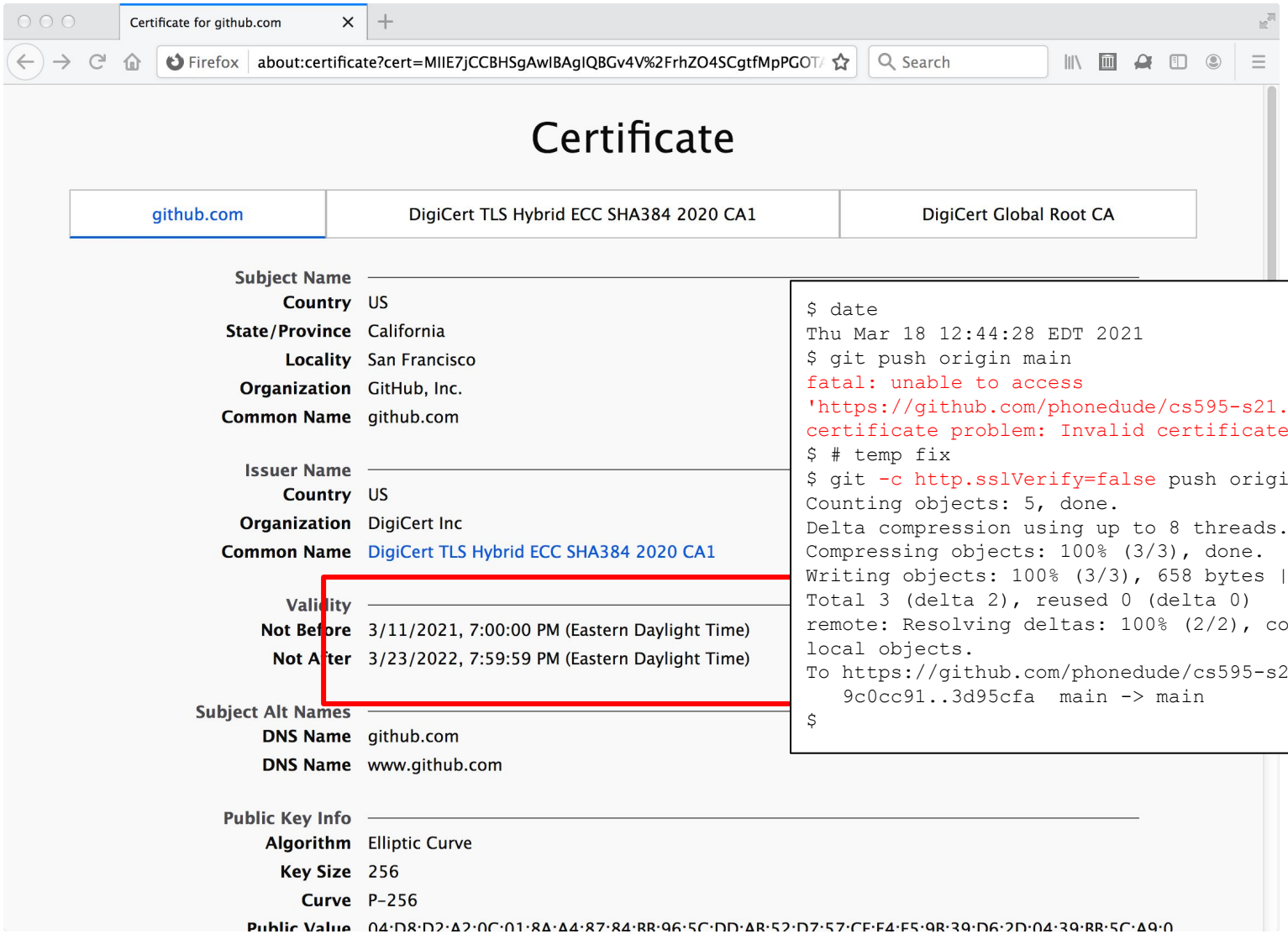
# Web Security

## Week 8 - Transport Layer Security

Old Dominion University
Department of Computer Science
CS 495/595 Spring 2022
Michael L. Nelson <mln@cs.odu.edu>

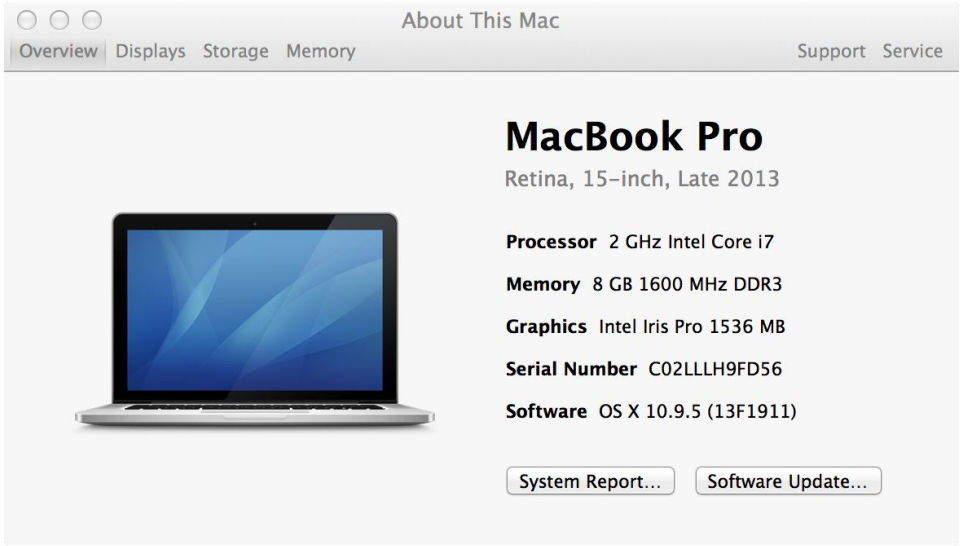
2022-03-14

# #irony




```
$ date
Thu Mar 18 12:44:28 EDT 2021
$ git push origin main
fatal: unable to access
'https://github.com/phonedude/cs595-s21.git/': SSL
certificate problem: Invalid certificate chain
$ # temp fix
$ git -c http.sslVerify=false push origin main
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 658 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2
local objects.
To https://github.com/phonedude/cs595-s21.git
   9c0cc91..3d95cfa  main -> main
$
```

# Ancient laptop = out of data CAs


more on this later…

# “HTTP” is not secure

- More and more, when we say “HTTP” we really mean “HTTPS”
- HTTP semantics (GET, POST, etc.) do not change, but in HTTPS all HTTP traffic “rides on top of” a secure layer

# HTTP request-response

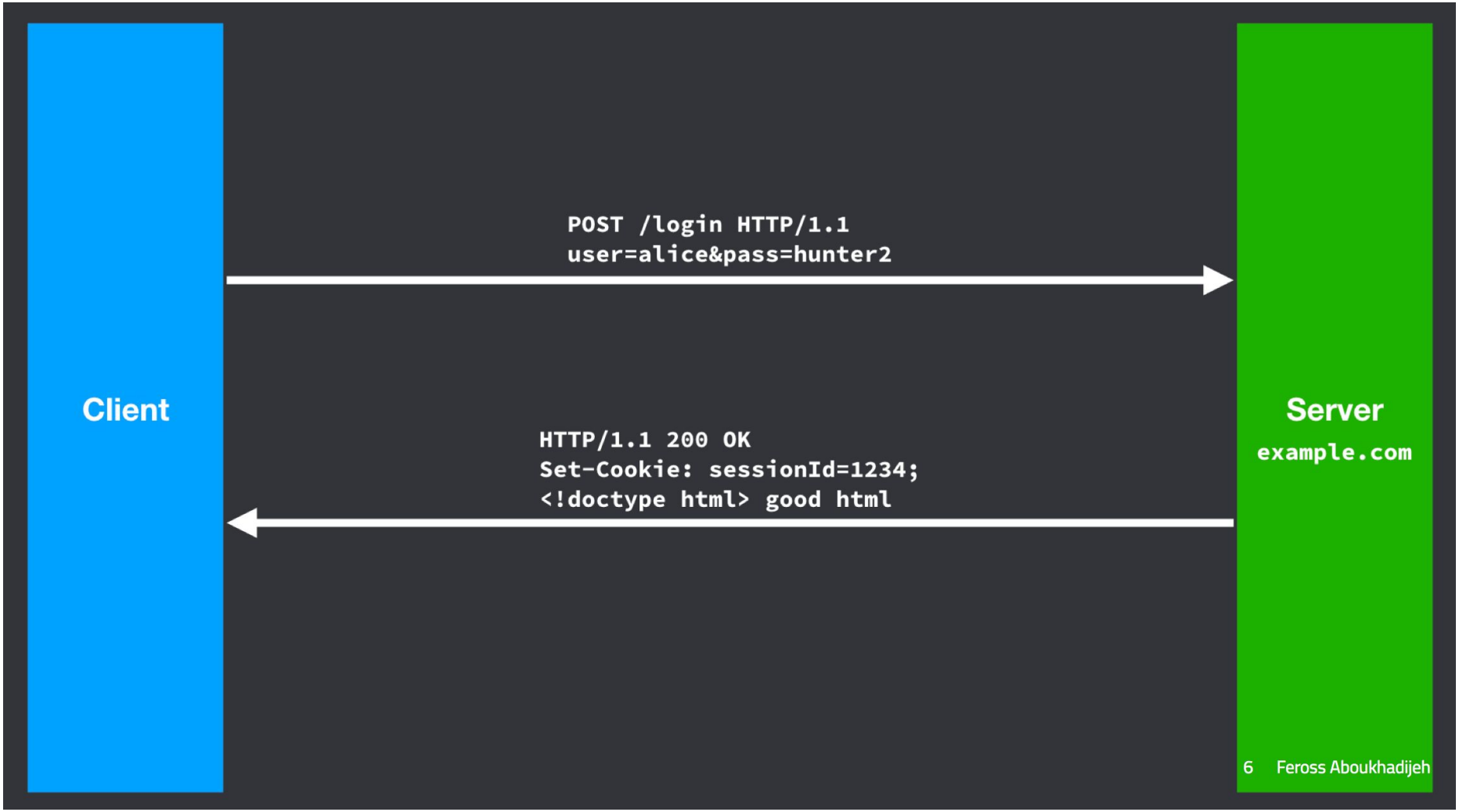
Client

Server

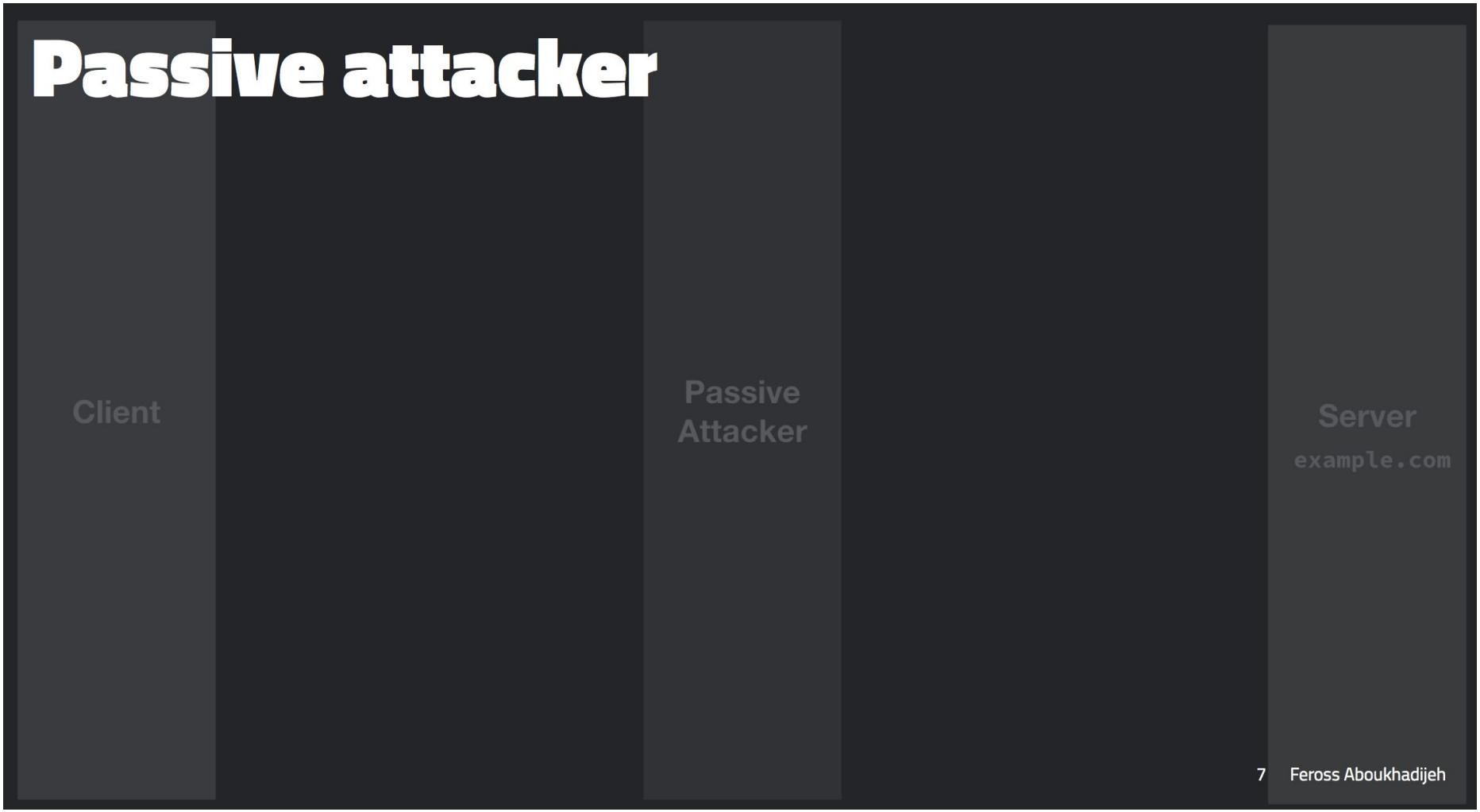
example.com

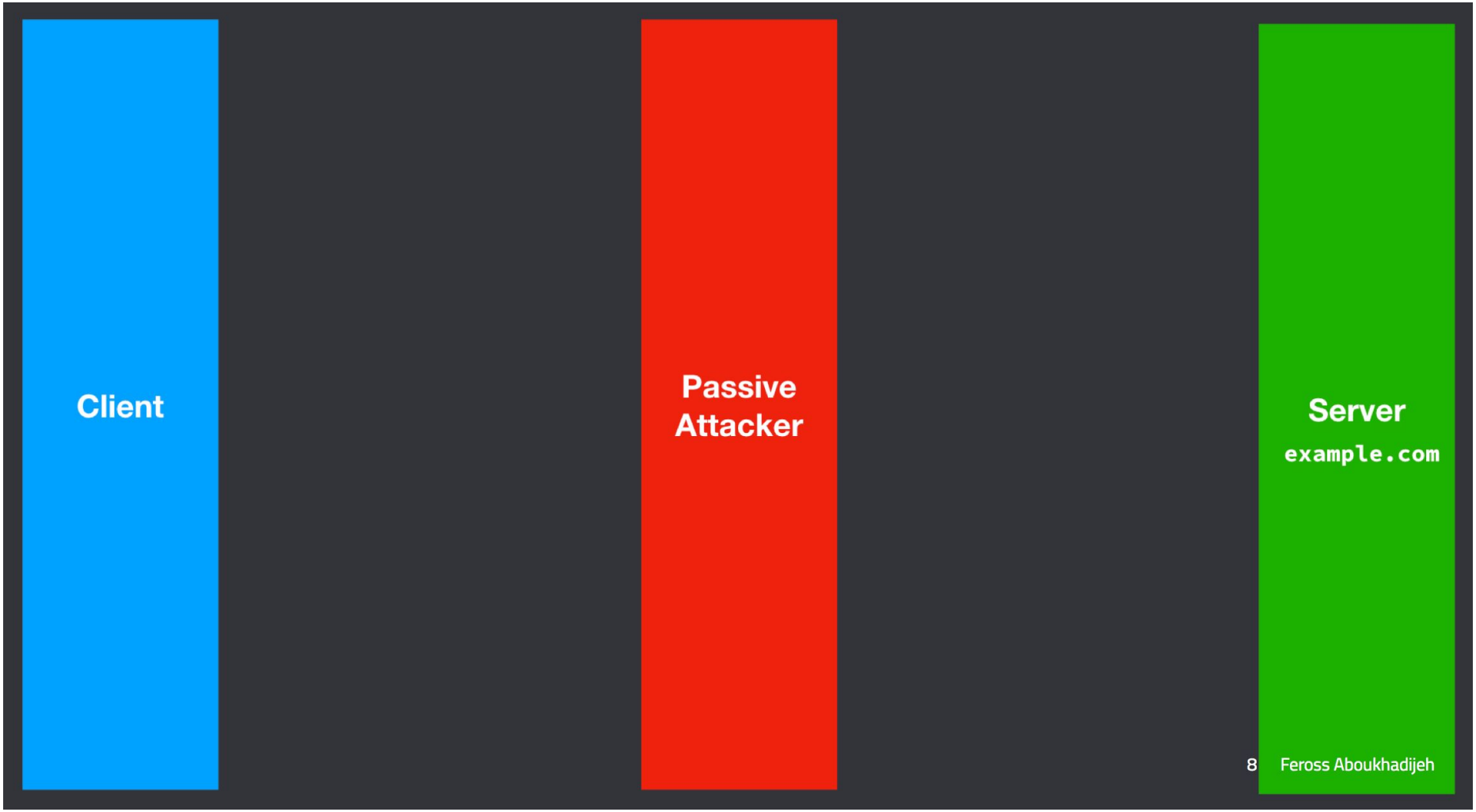

Client
Server
example.com
4
Feross Aboukhadijeh

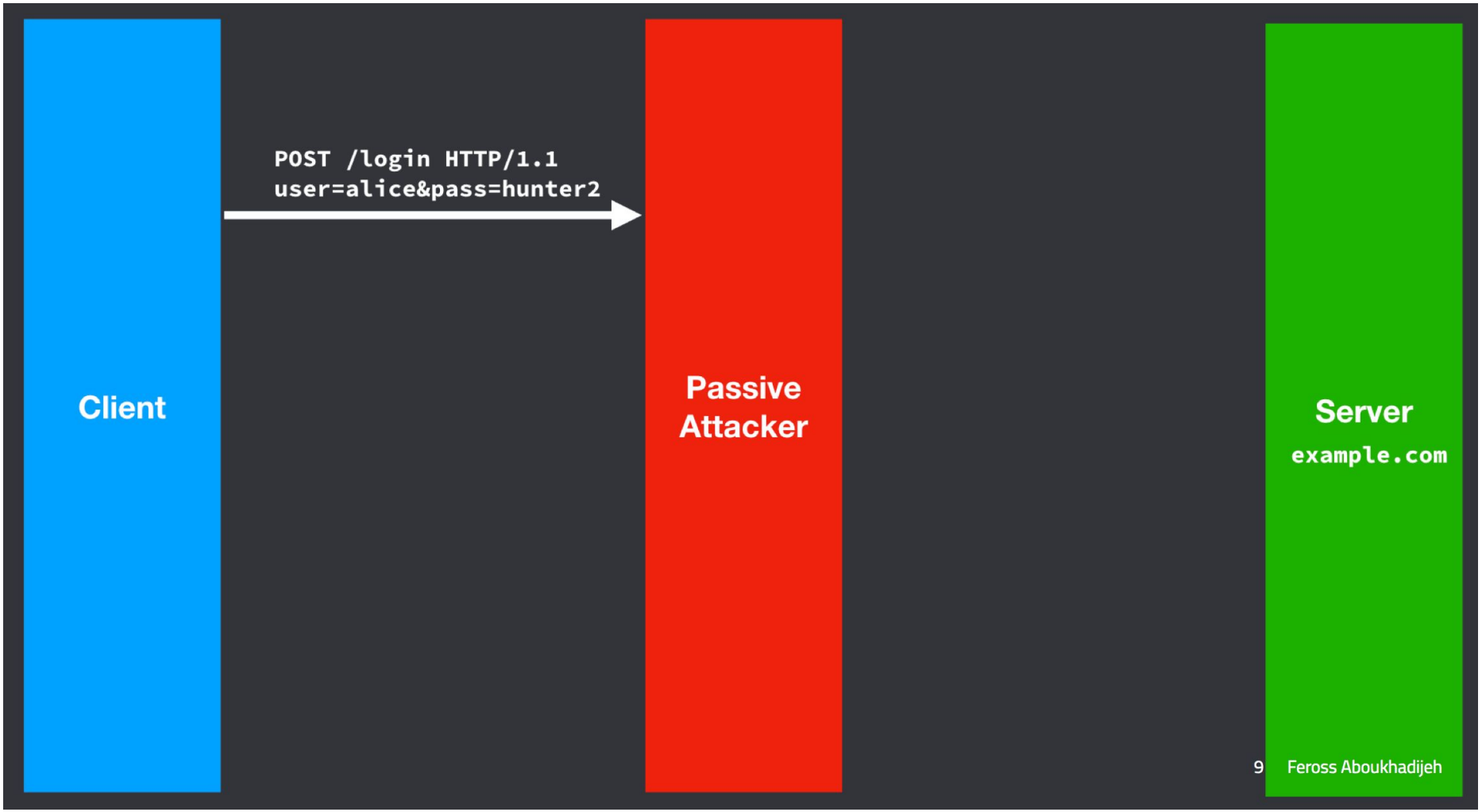

Client
POST /login HTTP/1.1
user=alice&pass=hunter2
Server
example.com
5 Feross Aboukhadijeh

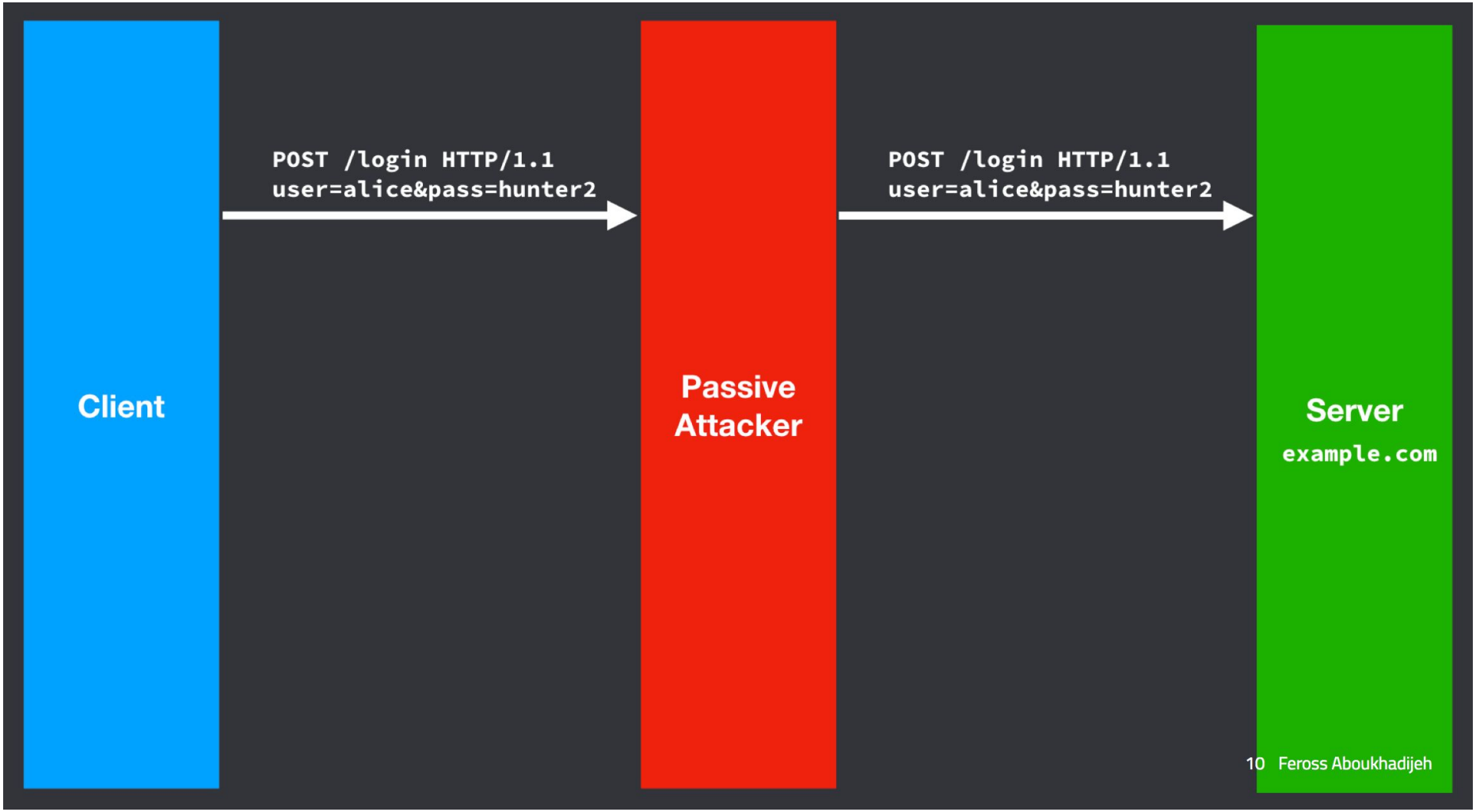

Client
POST /login HTTP/1.1
user=alice&pass=hunter2
HTTP/1.1 200 OK
Set-Cookie: sessionId=1234;
<!doctype html> good html
Server
example.com
6 Feross Aboukhadijeh

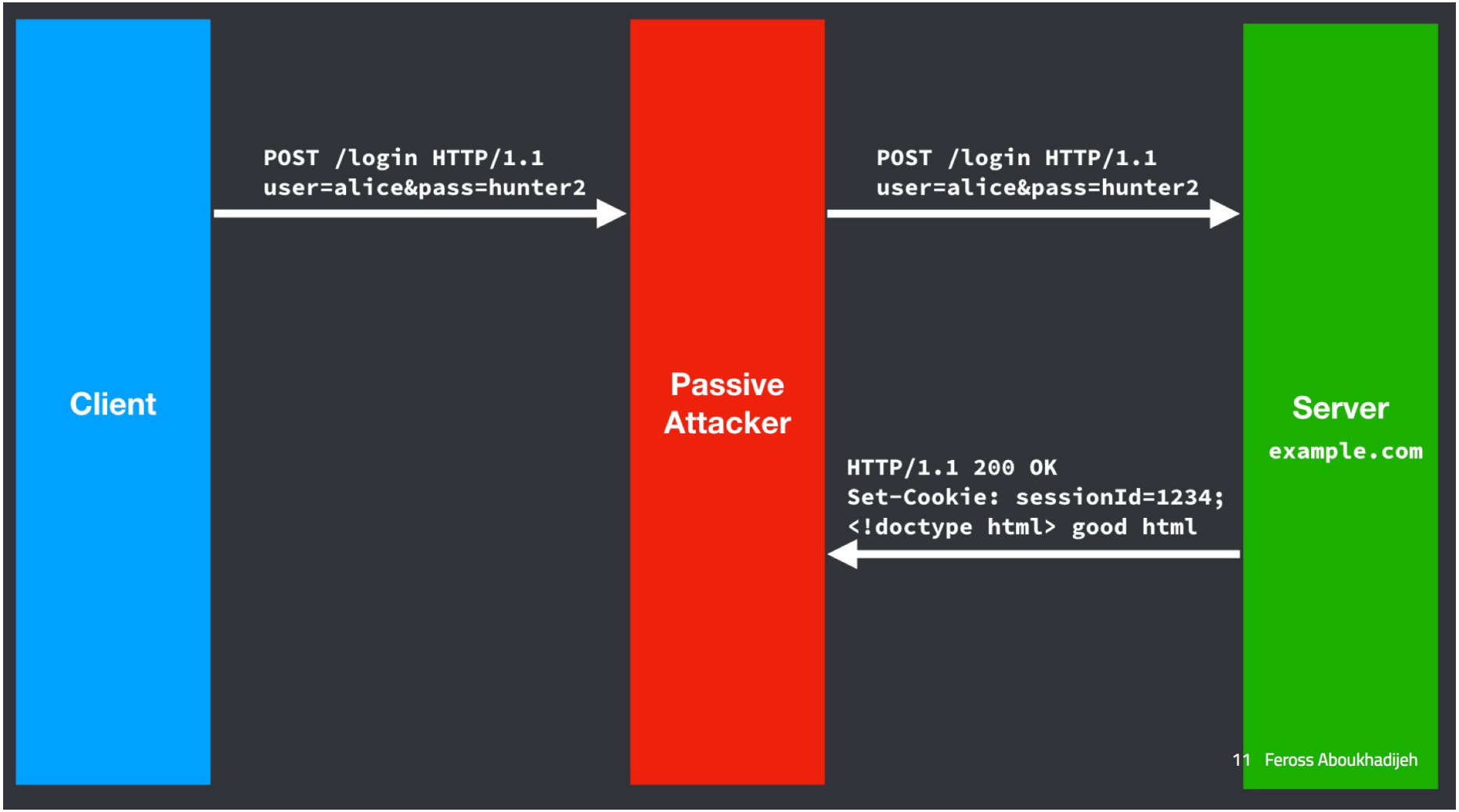
# Passive attacker

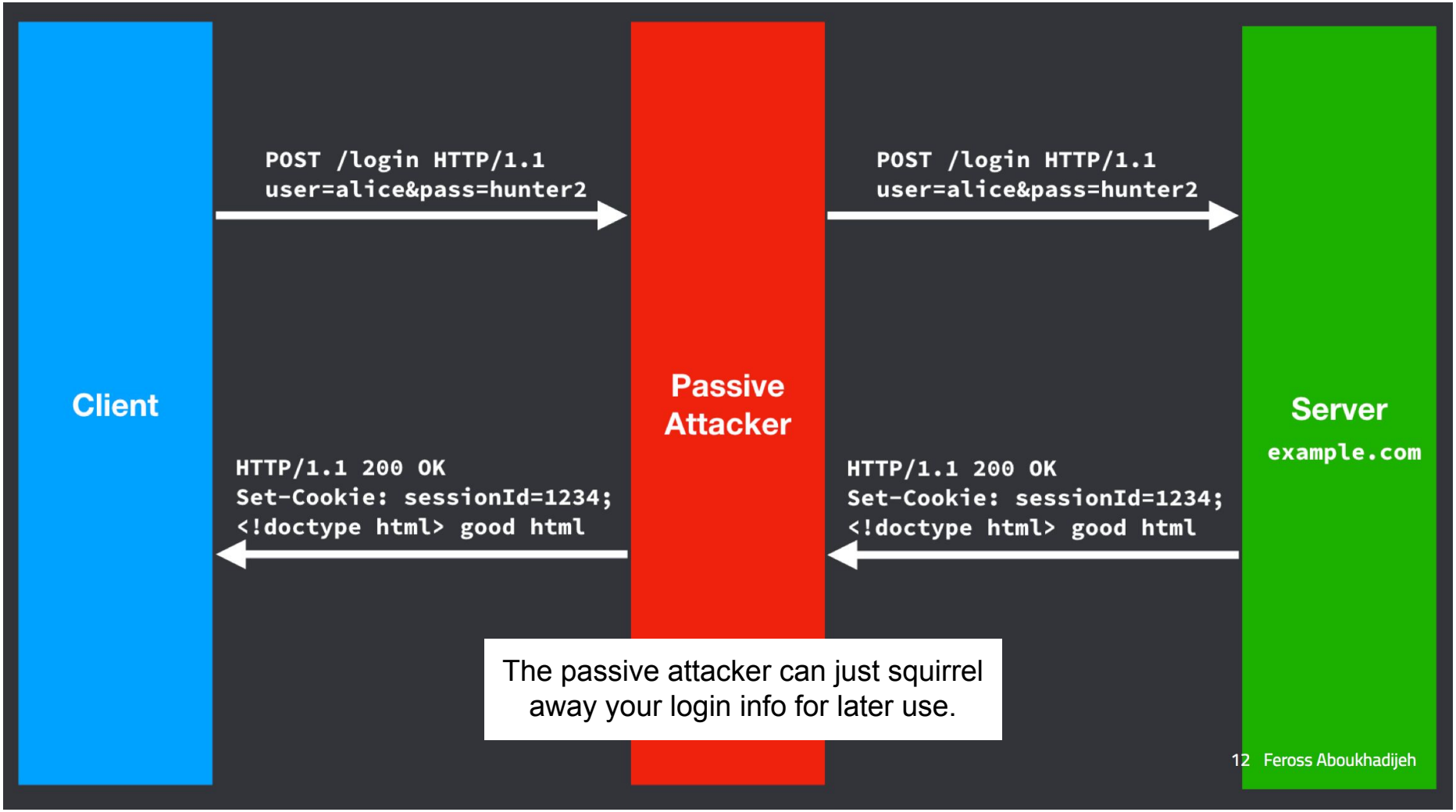


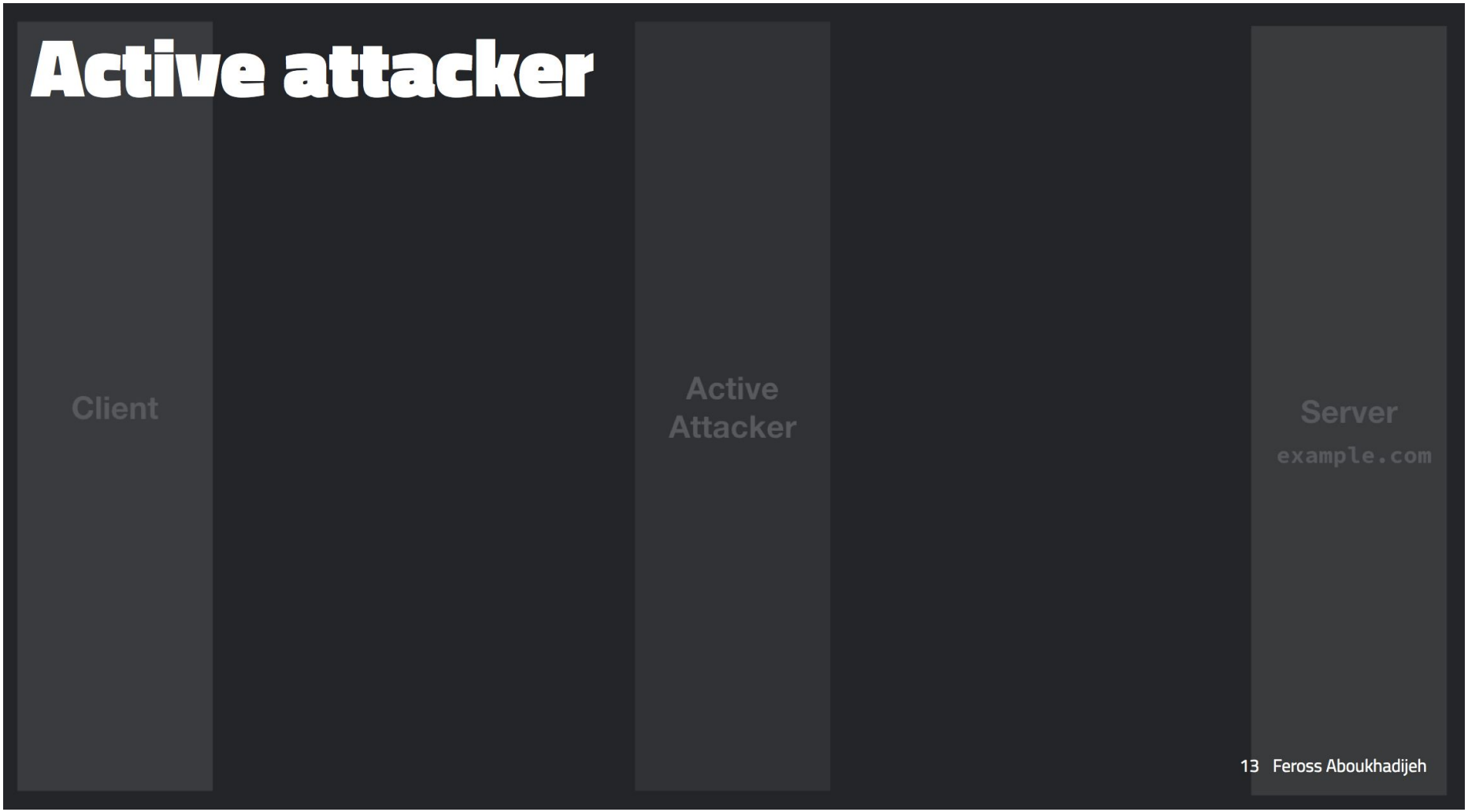

Client
Passive Attacker
Server
example.com
8
Feross Aboukhadijeh

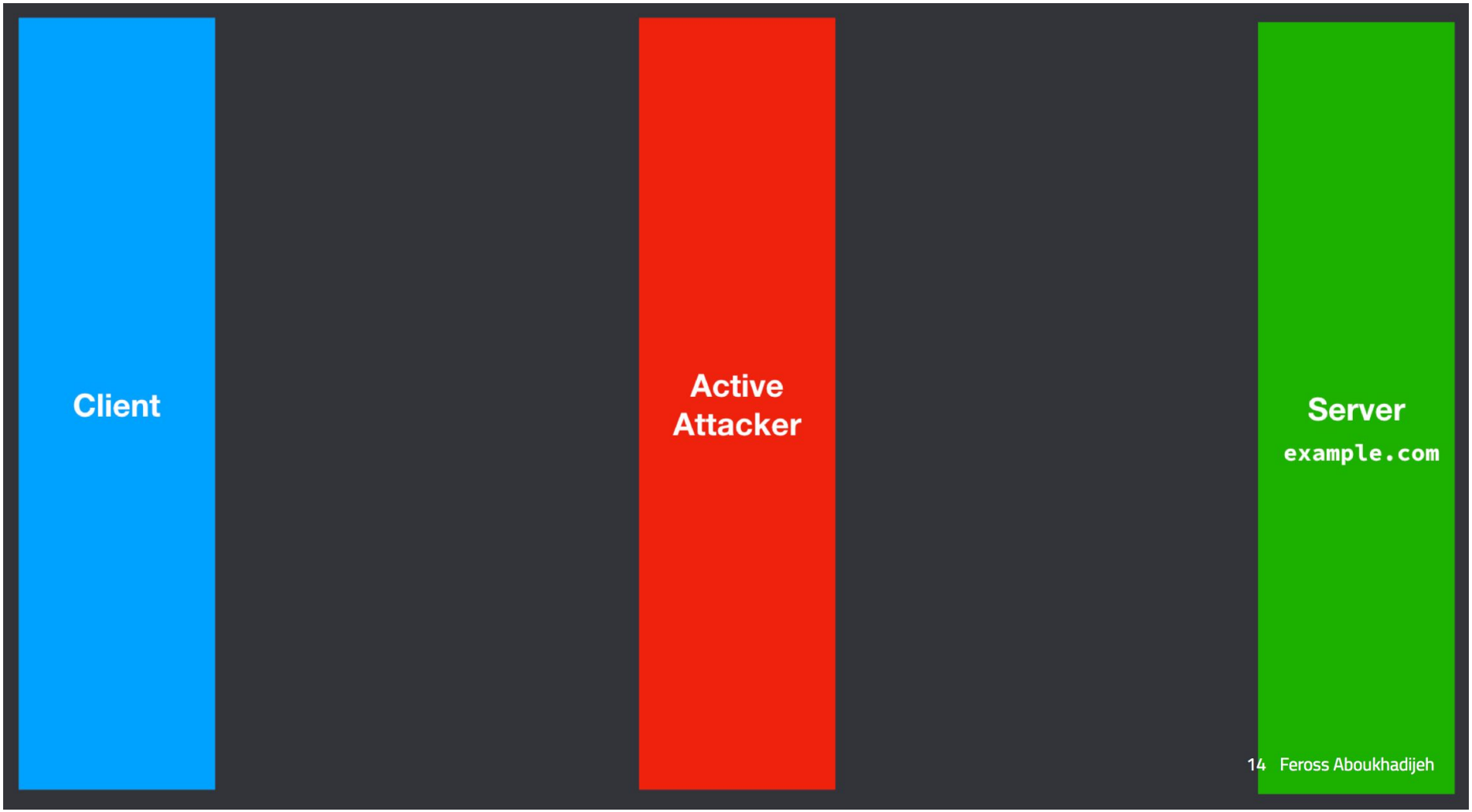

Client
POST /login HTTP/1.1
user=alice&pass=hunter2
Passive Attacker
Server
example.com
9
Feross Aboukhadijeh

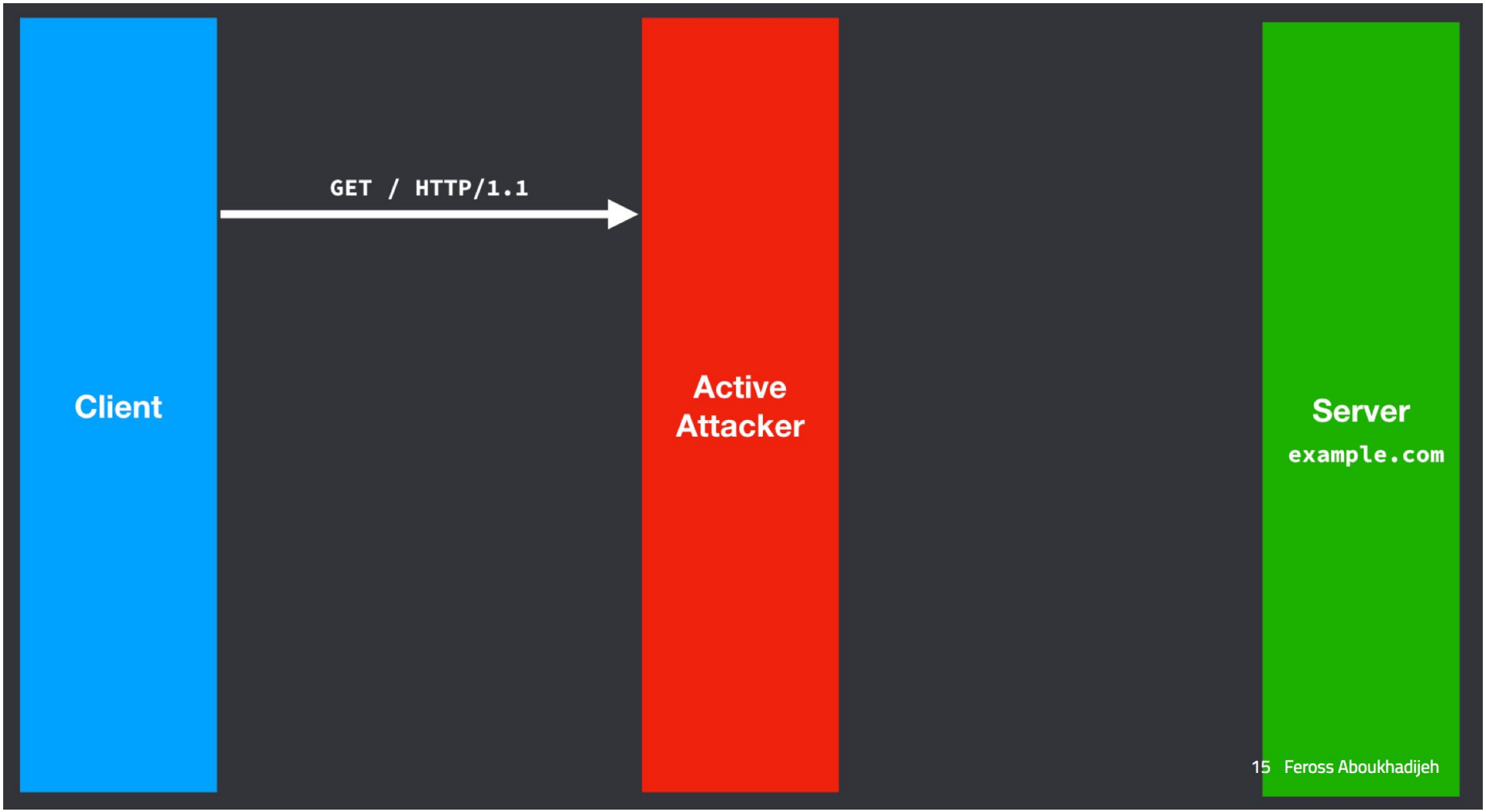

Client
POST /login HTTP/1.1
user=alice&pass=hunter2
Passive Attacker
POST /login HTTP/1.1
user=alice&pass=hunter2
Server
example.com
10 Feross Aboukhadijeh


Client
POST /login HTTP/1.1
user=alice&pass=hunter2
Passive Attacker
POST /login HTTP/1.1
user=alice&pass=hunter2
Server
example.com
HTTP/1.1 200 OK
Set-Cookie: sessionId=1234;
<!doctype html> good html
11 Feross Aboukhadijeh

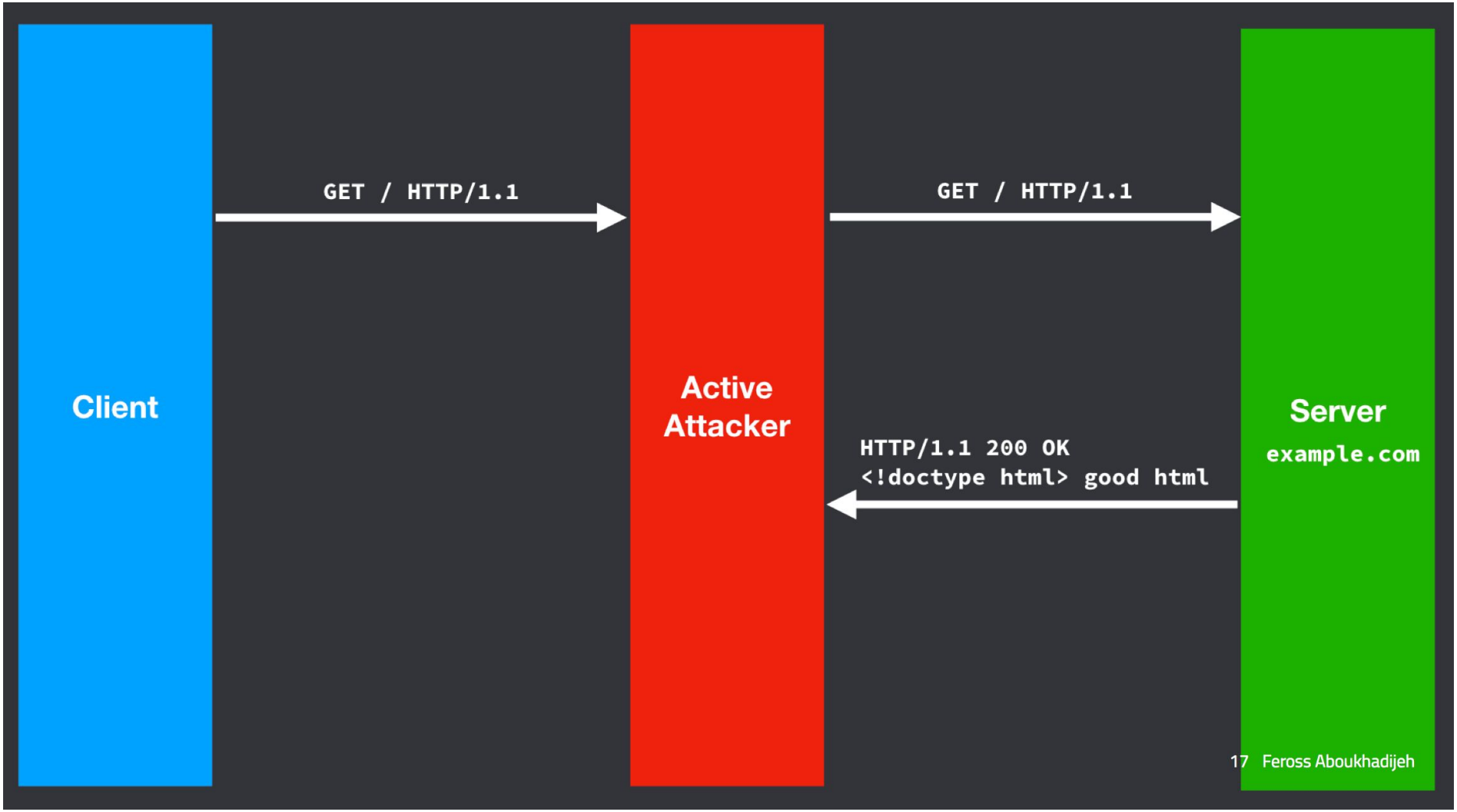

Client
POST /login HTTP/1.1
user=alice&pass=hunter2
Passive Attacker
POST /login HTTP/1.1
user=alice&pass=hunter2
Server
example.com
HTTP/1.1 200 OK
Set-Cookie: sessionId=1234;
<!doctype html> good html
HTTP/1.1 200 OK
Set-Cookie: sessionId=1234;
<!doctype html> good html
The passive attacker can just squirrel away your login info for later use.
12 Feross Aboukhadijeh

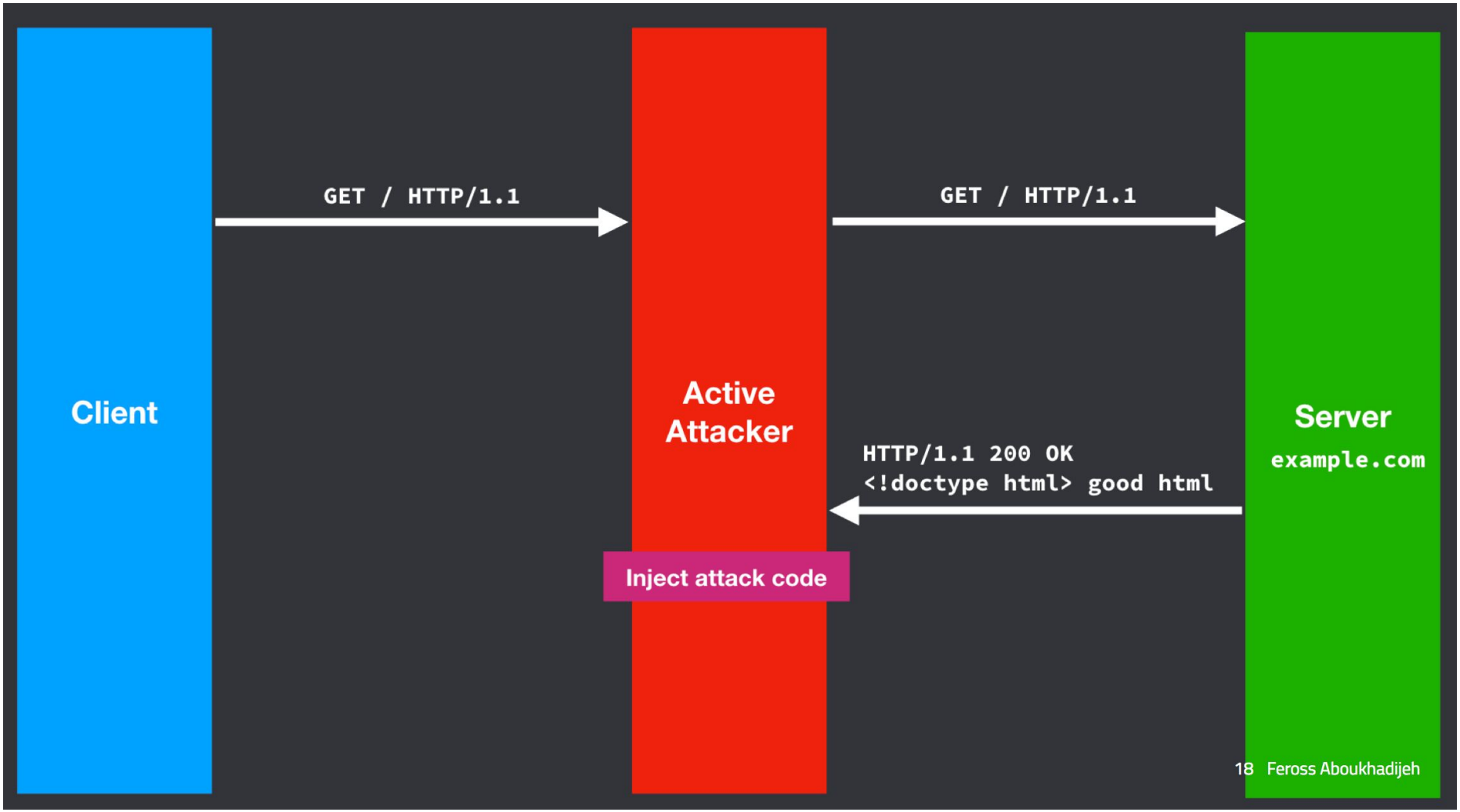

Active attacker
Client
Active Attacker
Server
example.com
13 Feross Aboukhadijeh

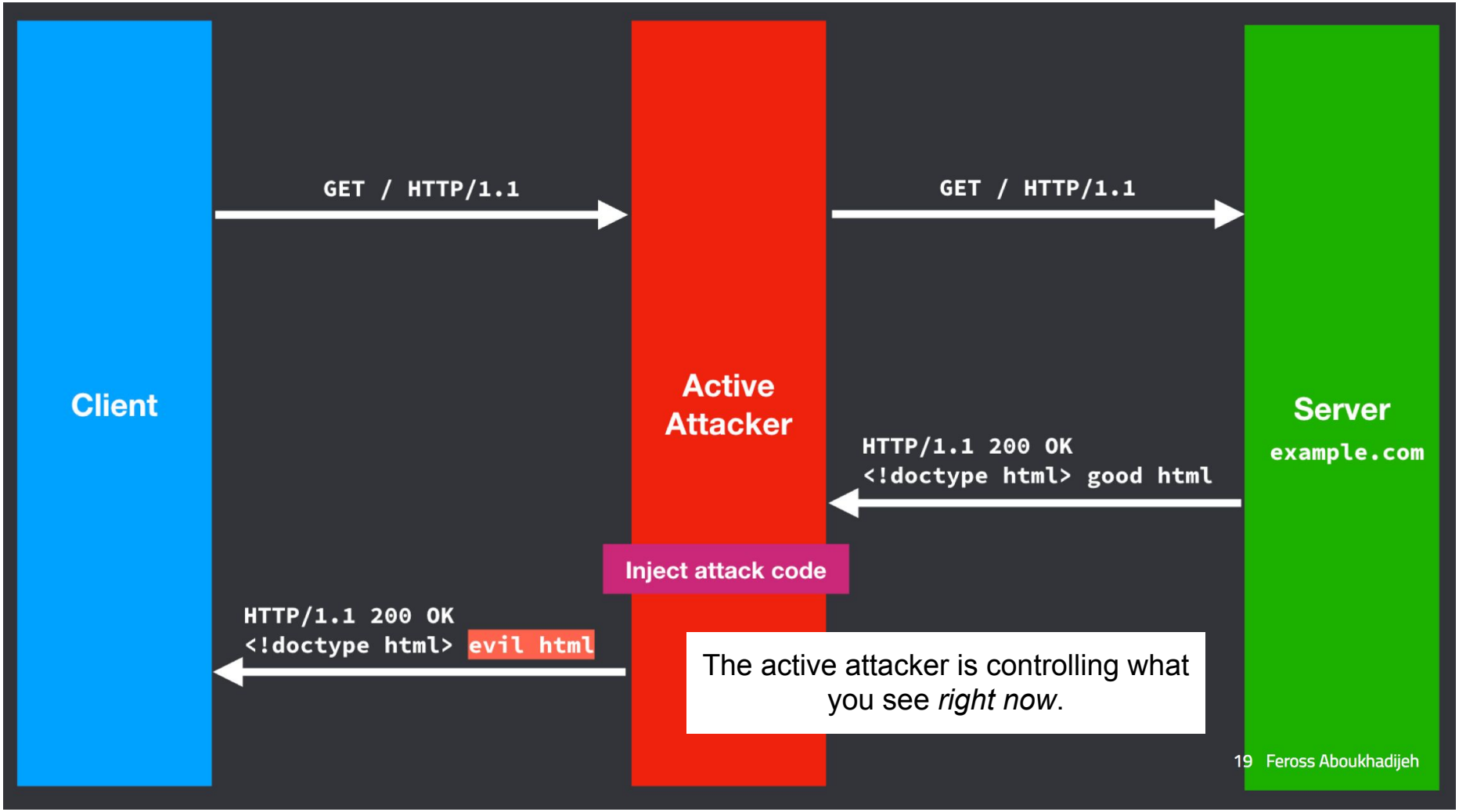

Client
Active Attacker
Server
example.com
14 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
Server
example.com
15 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
16 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 200 OK
<!doctype html> good html
17 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 200 OK
<!doctype html> good html
Inject attack code
18 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 200 OK
<!doctype html> good html
Inject attack code
HTTP/1.1 200 OK
<!doctype html> evil html
The active attacker is controlling what you see *right now*.
19 Feross Aboukhadijeh

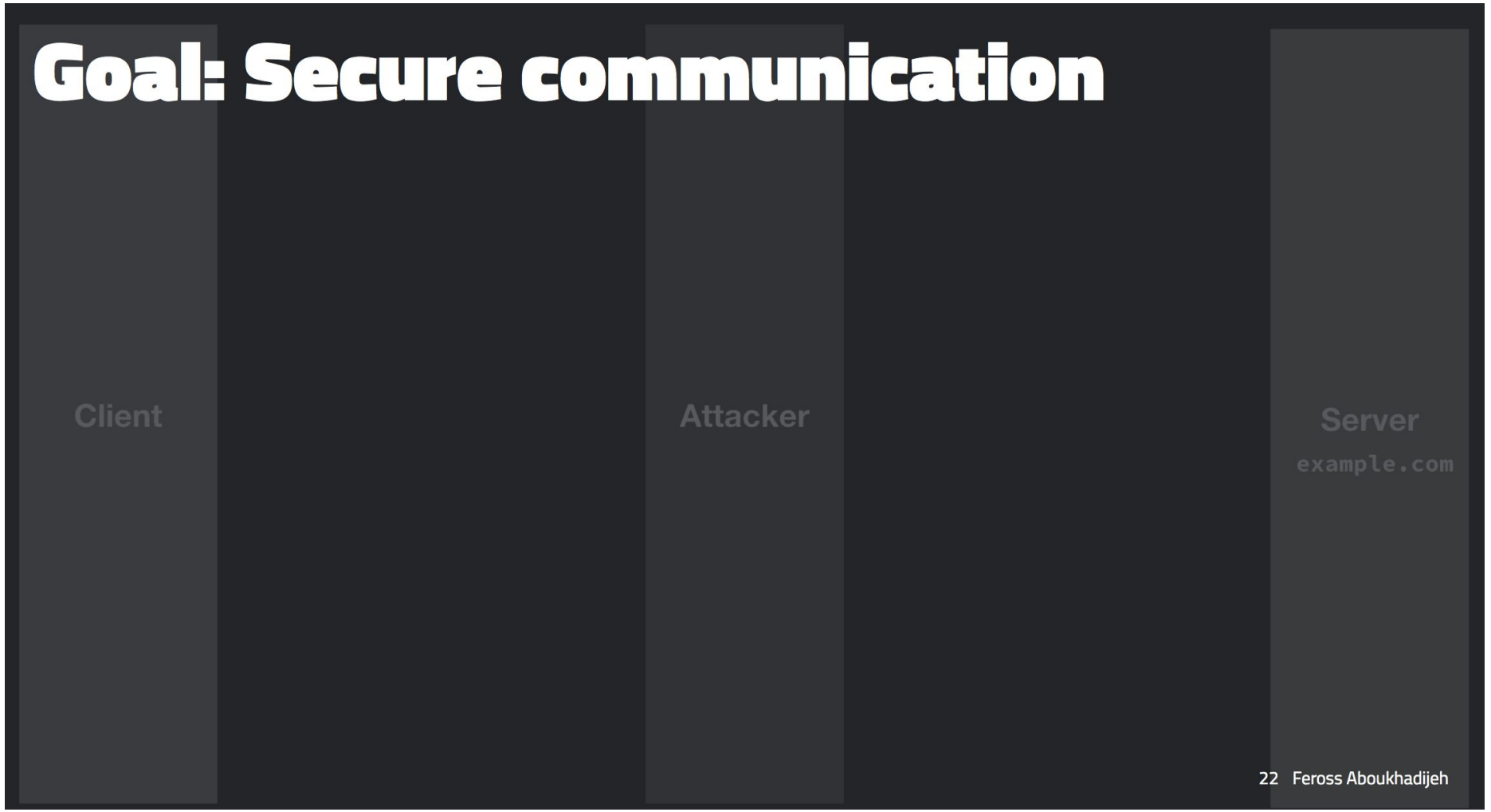
# What is the threat model?

- Network attackers control network infrastructure like routers or DNS servers
- Network attackers may eavesdrop, inject, block, or modify packets
- Potential network attackers occur anywhere there is an untrusted router or ISP
  - Wireless networks at cafes or hotels
  - Border gateways between countries

# Goal: Secure communications

- Secure communication requires three properties
  - Privacy: No eavesdropping
  - Integrity: No tampering
  - Authentication: No impersonation

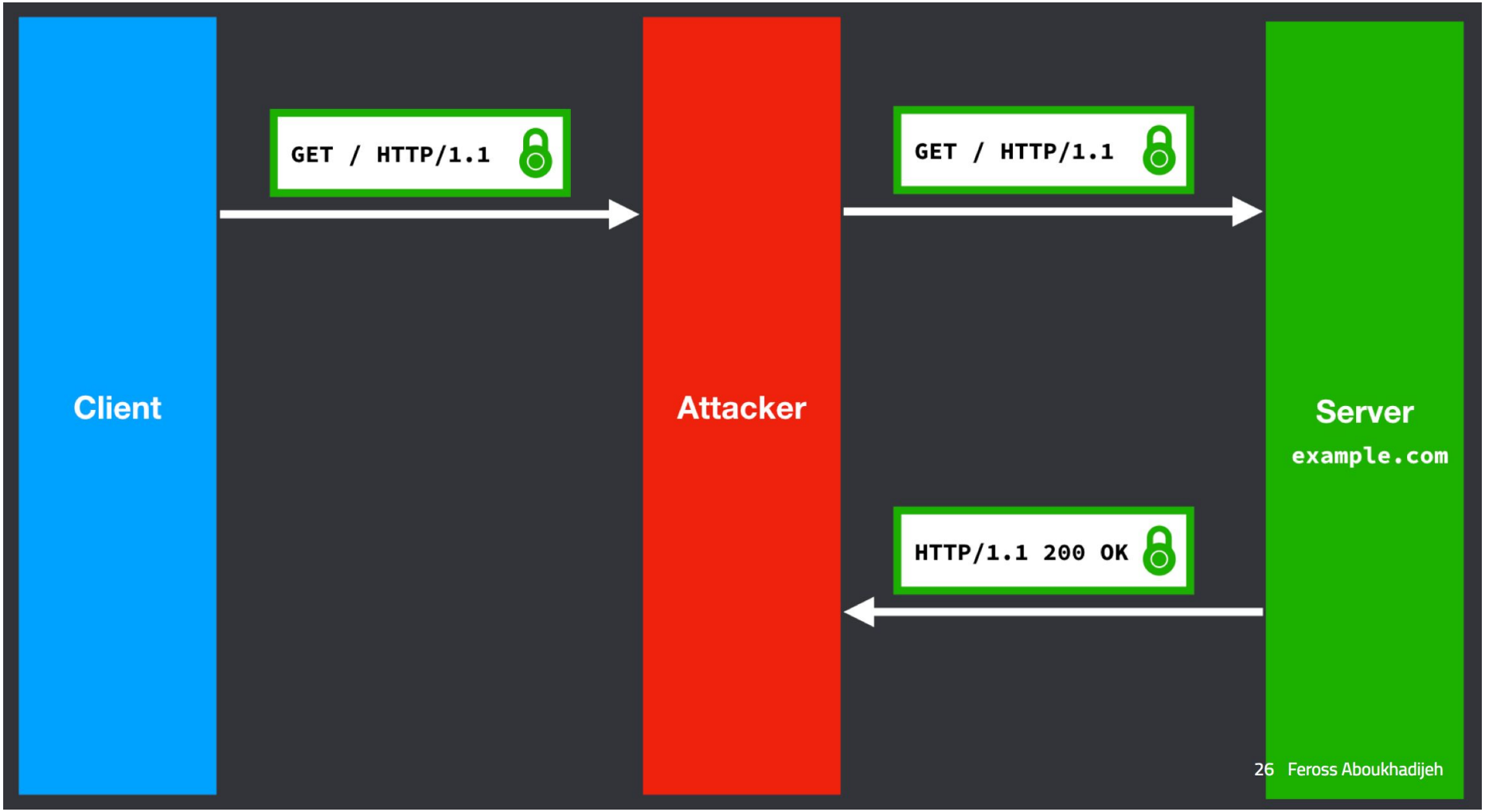
# Goal: Secure communication

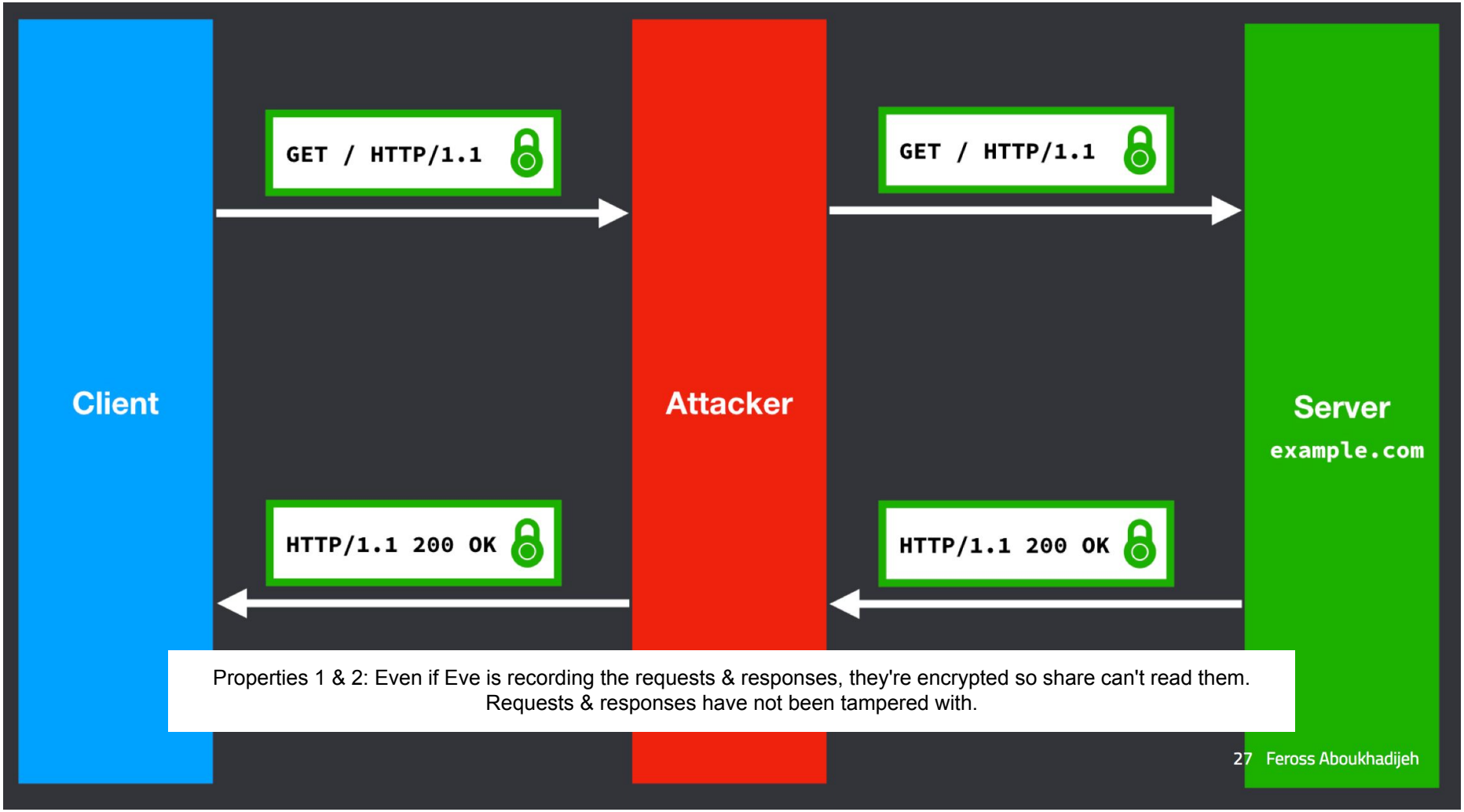


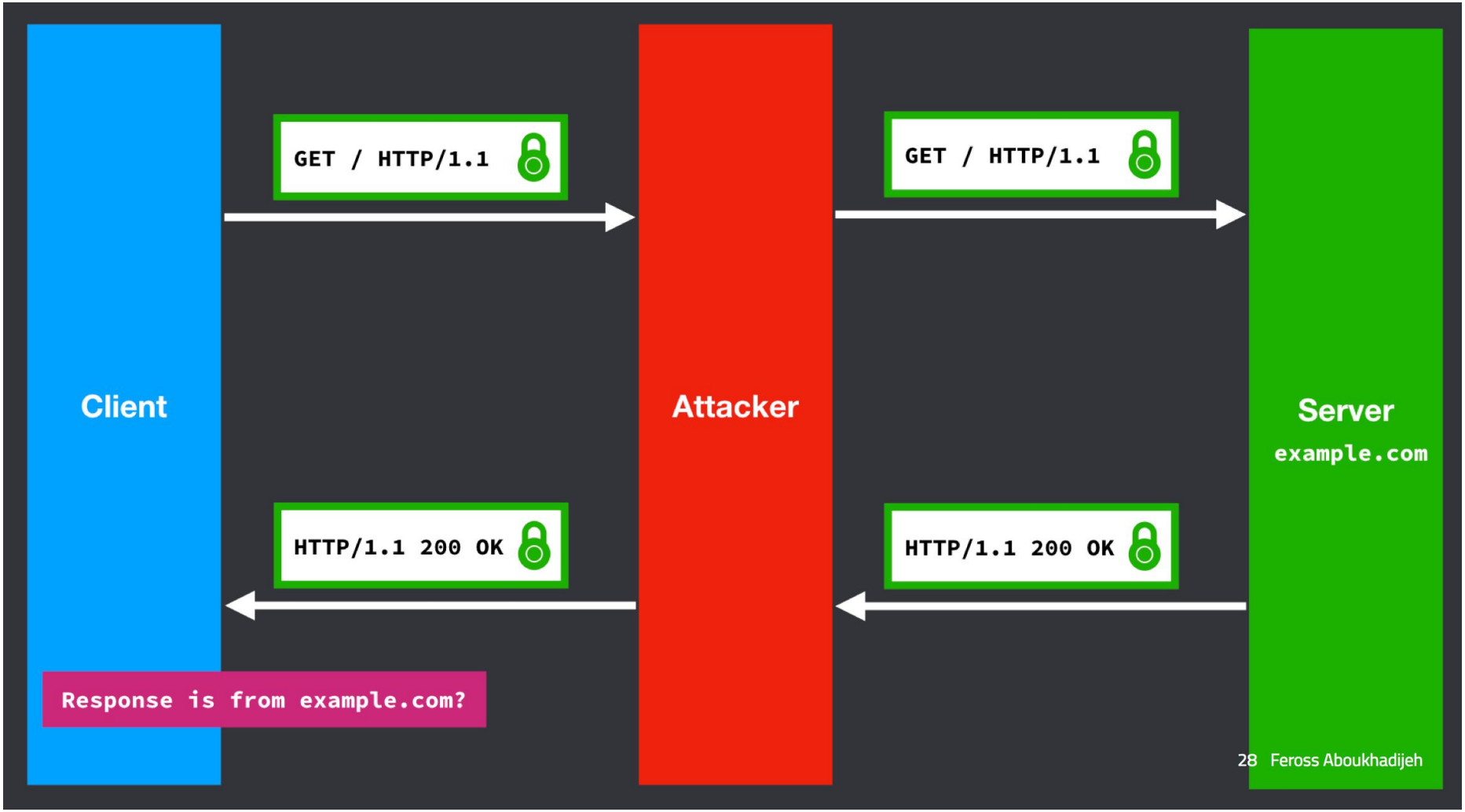

Client
Attacker
Server
example.com
23 Feross Aboukhadijeh

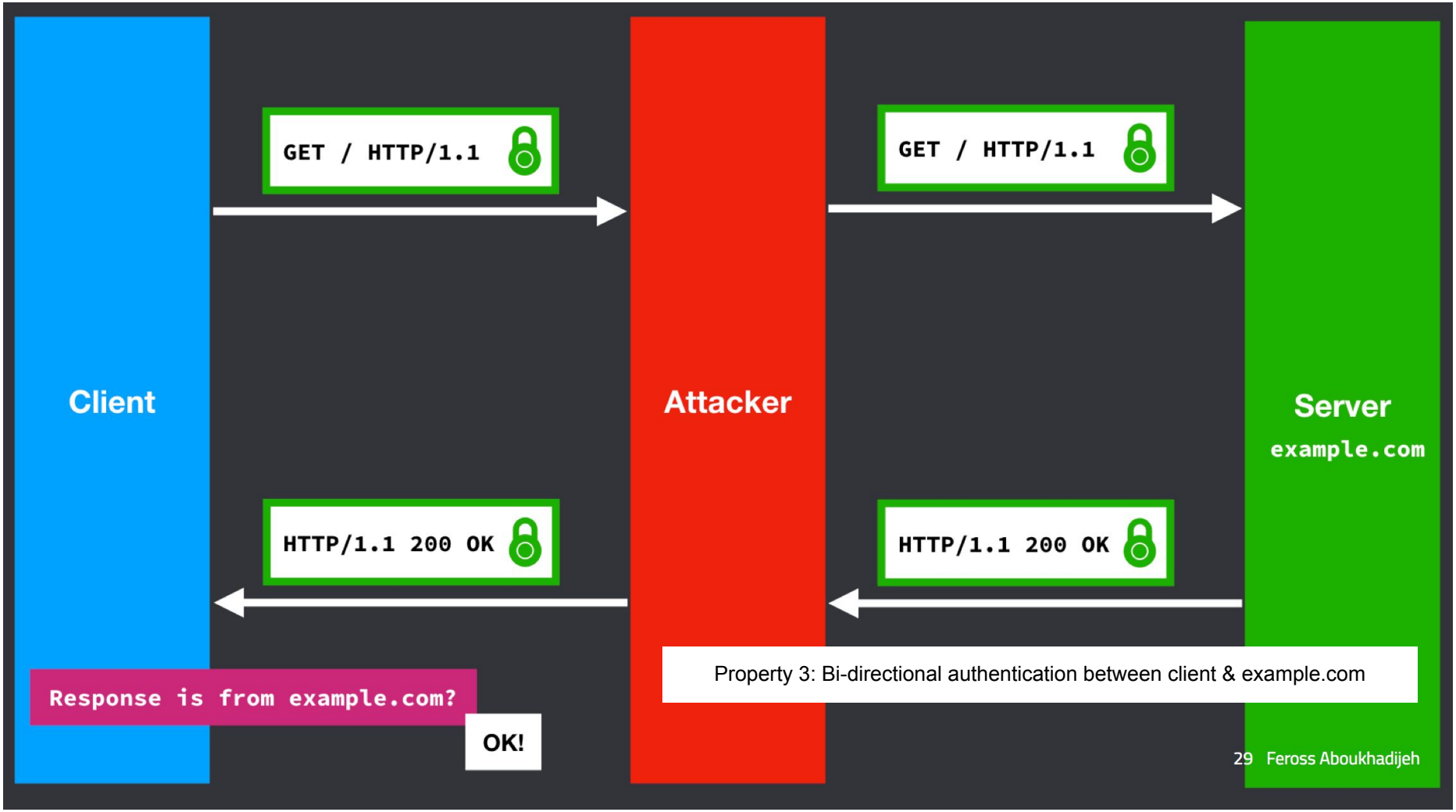

Client
GET / HTTP/1.1
Attacker
Server
example.com
24 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Attacker
GET / HTTP/1.1
Server
example.com
25 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 200 OK
26 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 200 OK
HTTP/1.1 200 OK
Properties 1 & 2: Even if Eve is recording the requests & responses, they're encrypted so share can't read them.
Requests & responses have not been tampered with.
27 Feross Aboukhadijeh


Client
Attacker
Server
example.com
GET / HTTP/1.1
GET / HTTP/1.1
HTTP/1.1 200 OK
HTTP/1.1 200 OK
Response is from example.com?
28 Feross Aboukhadijeh


Client
Attacker
Server
example.com
GET / HTTP/1.1
GET / HTTP/1.1
HTTP/1.1 200 OK
HTTP/1.1 200 OK
Response is from example.com?
OK!
Property 3: Bi-directional authentication between client & example.com
29 Feross Aboukhadijeh

# Transport Layer Security (TLS)

- Hypertext Transfer Protocol Secure (HTTPS) keeps browsing safe by securely connecting the browser with the website server
- HTTPS relies on Transport Layer Security (TLS) encryption to secure connections
- TLS is used with web traffic, email, instant messaging, voice over IP (VoIP), and many other protocols
  - When TLS is used with HTTP, we call it HTTPS

# TLS is the successor to SSL, but the terms are often used interchangeably

**SSL and TLS protocols**

| Protocol | Published | Status |
|---|---|---|
| **SSL 1.0** | Unpublished | Unpublished |
| **SSL 2.0** | 1995 | Deprecated in 2011 (RFC 6176) |
| **SSL 3.0** | 1996 | Deprecated in 2015 (RFC 7568) |
| **TLS 1.0** | 1999 | Deprecated in 2020 (RFC 8996)[9][10][11] |
| **TLS 1.1** | 2006 | Deprecated in 2020 (RFC 8996)[9][10][11] |
| **TLS 1.2** | 2008 | |
| **TLS 1.3** | 2018 | |

https://en.wikipedia.org/wiki/Transport_Layer_Security

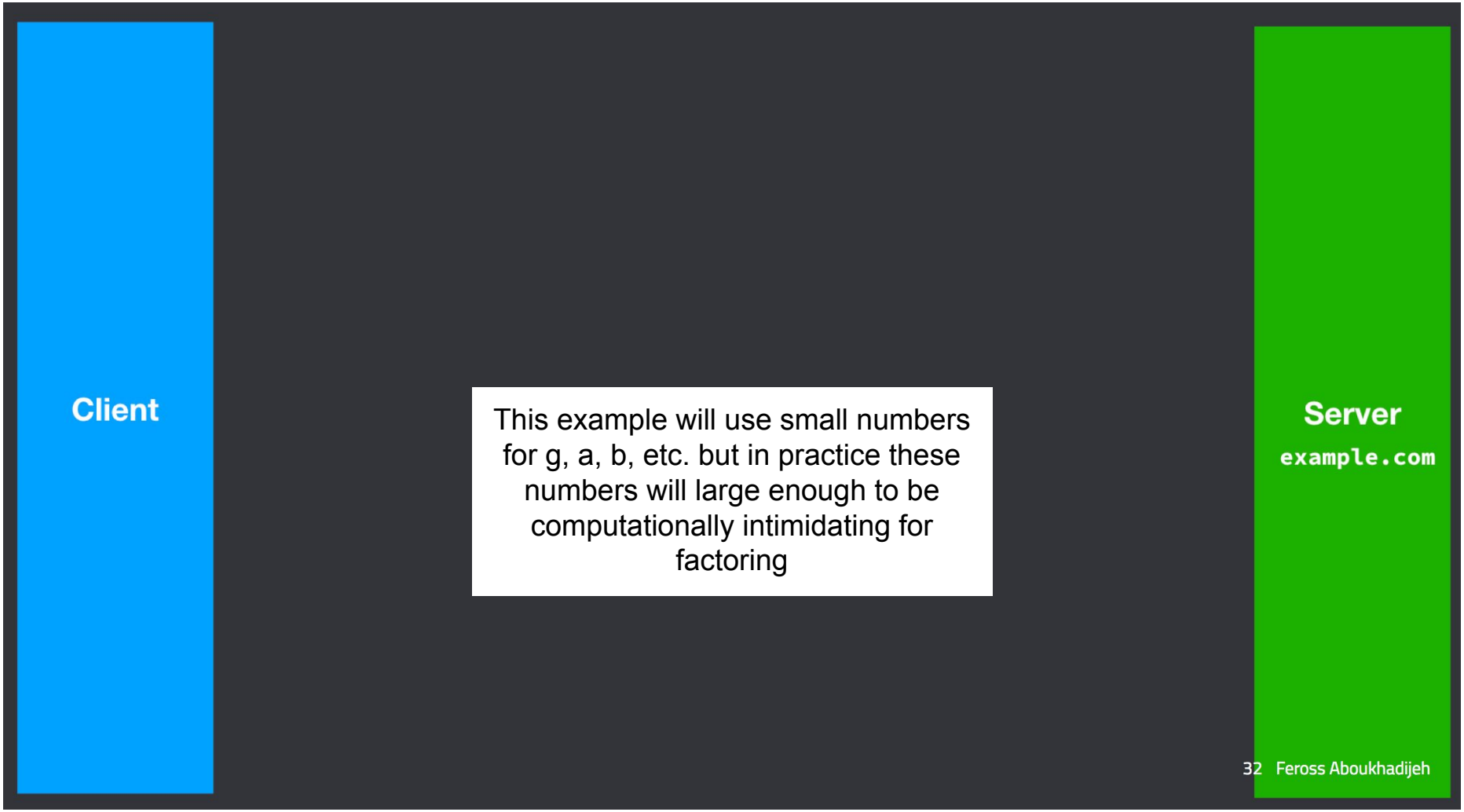
# Anonymous Diffie-Hellman key exchange

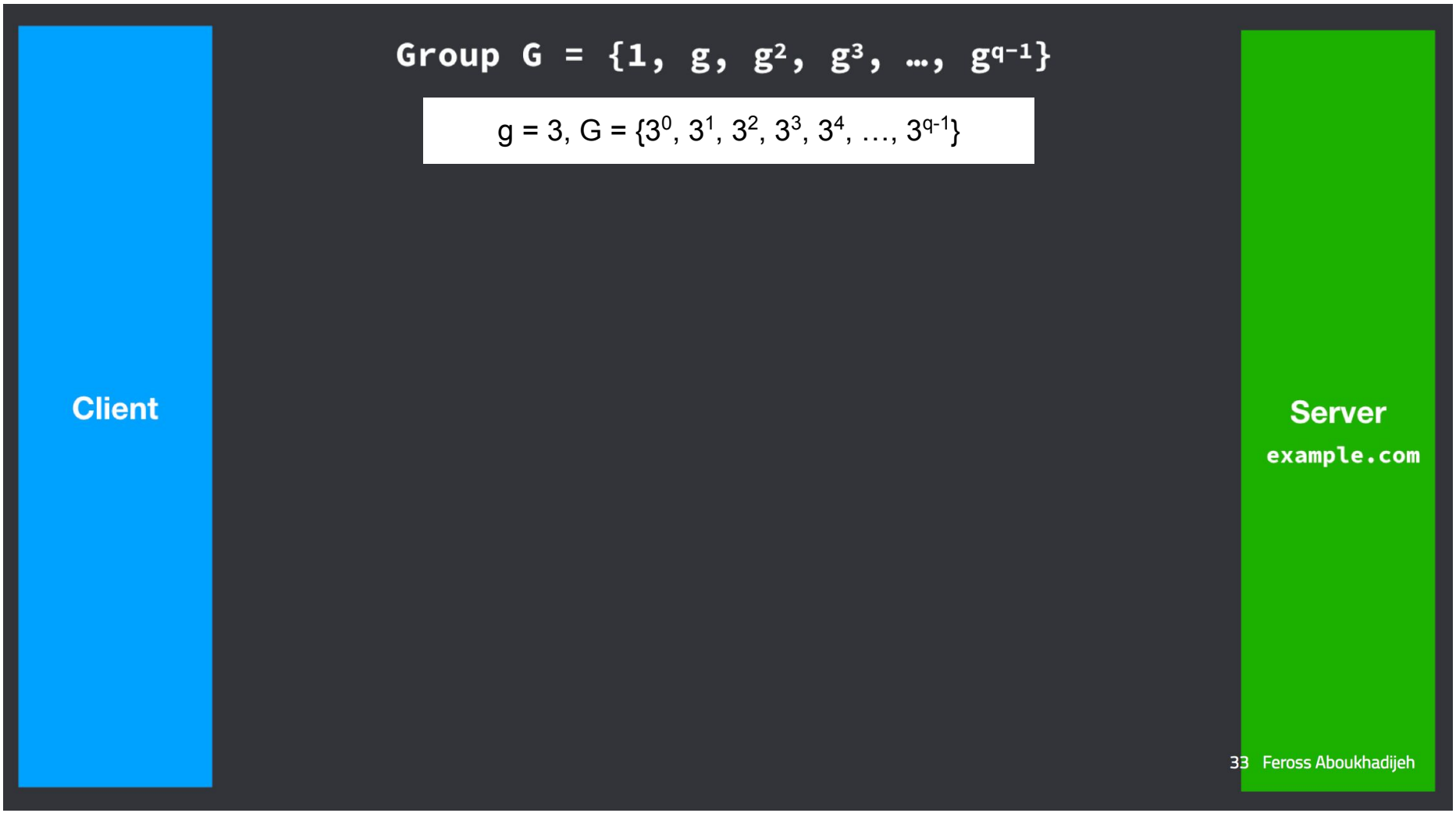
Client

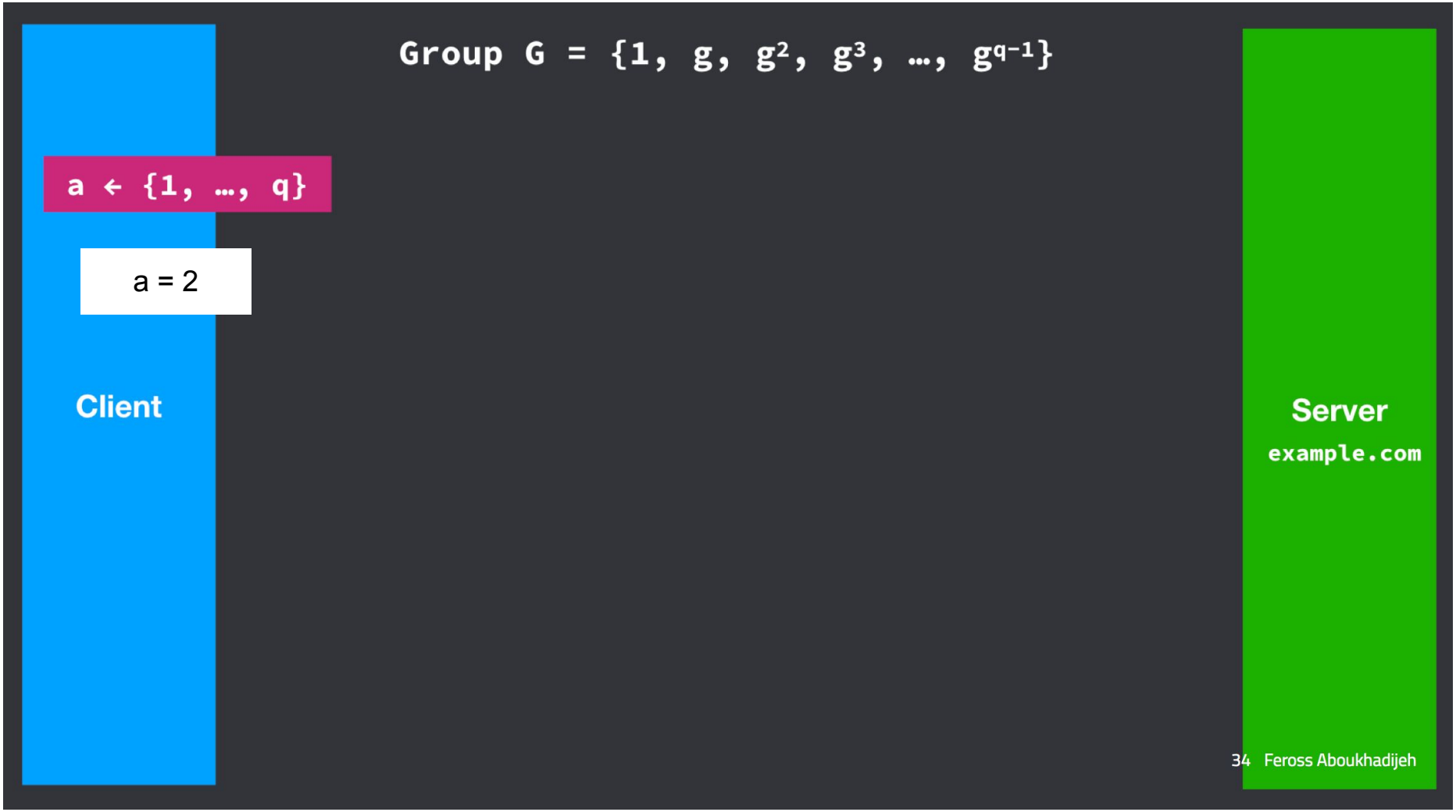
Protects against passive attackers,
but not active attackers.

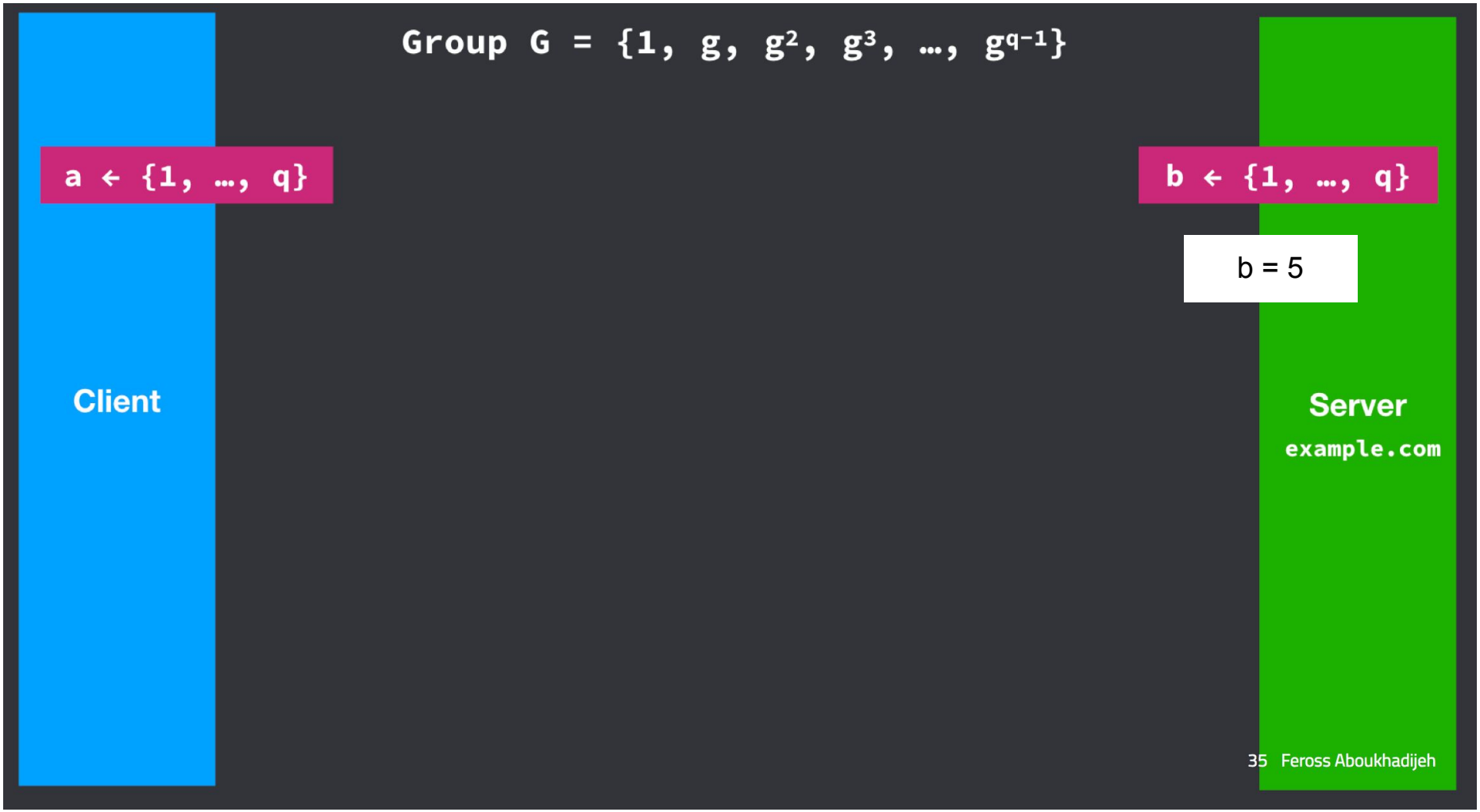
Server
example.com

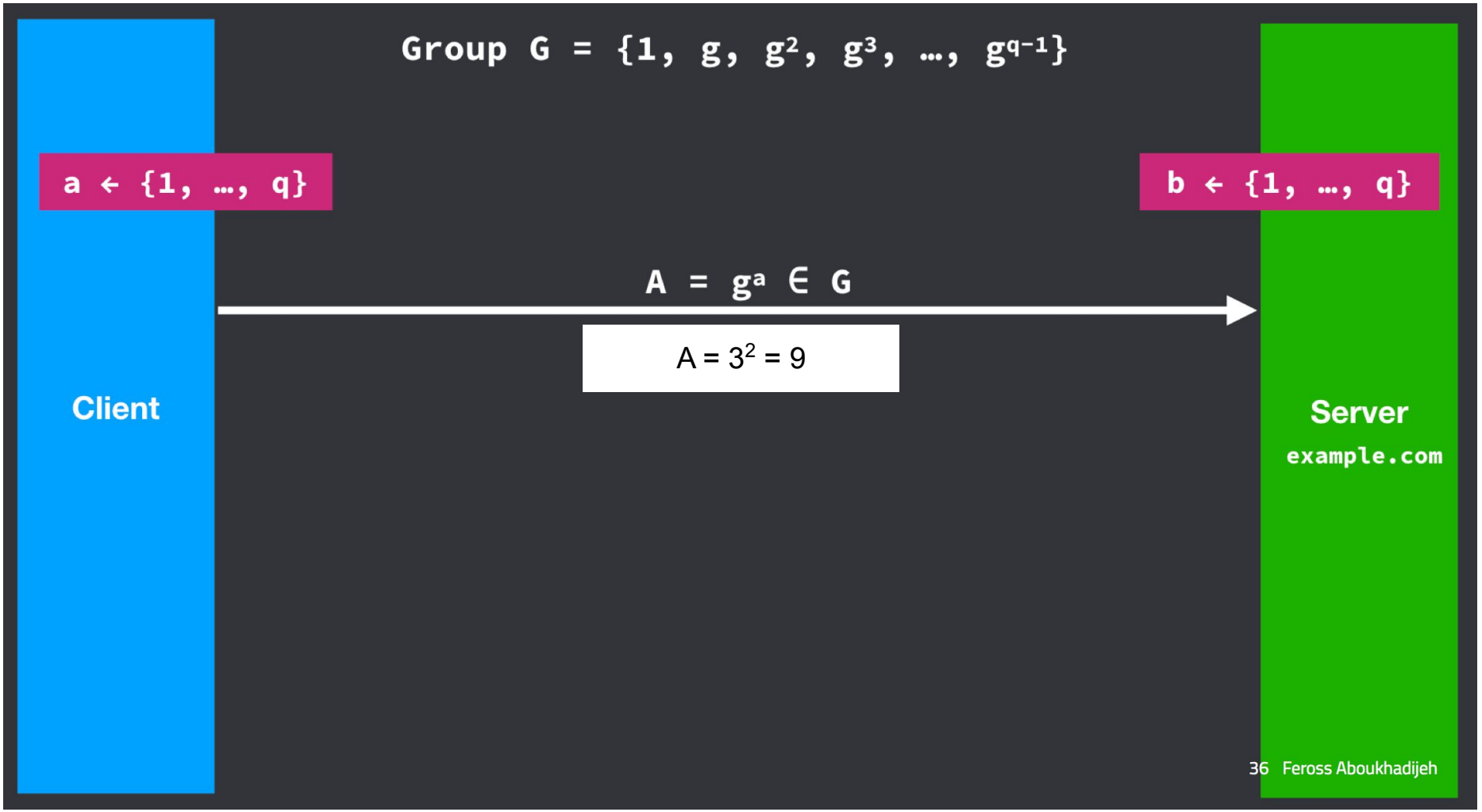
31 Feross Aboukhadijeh

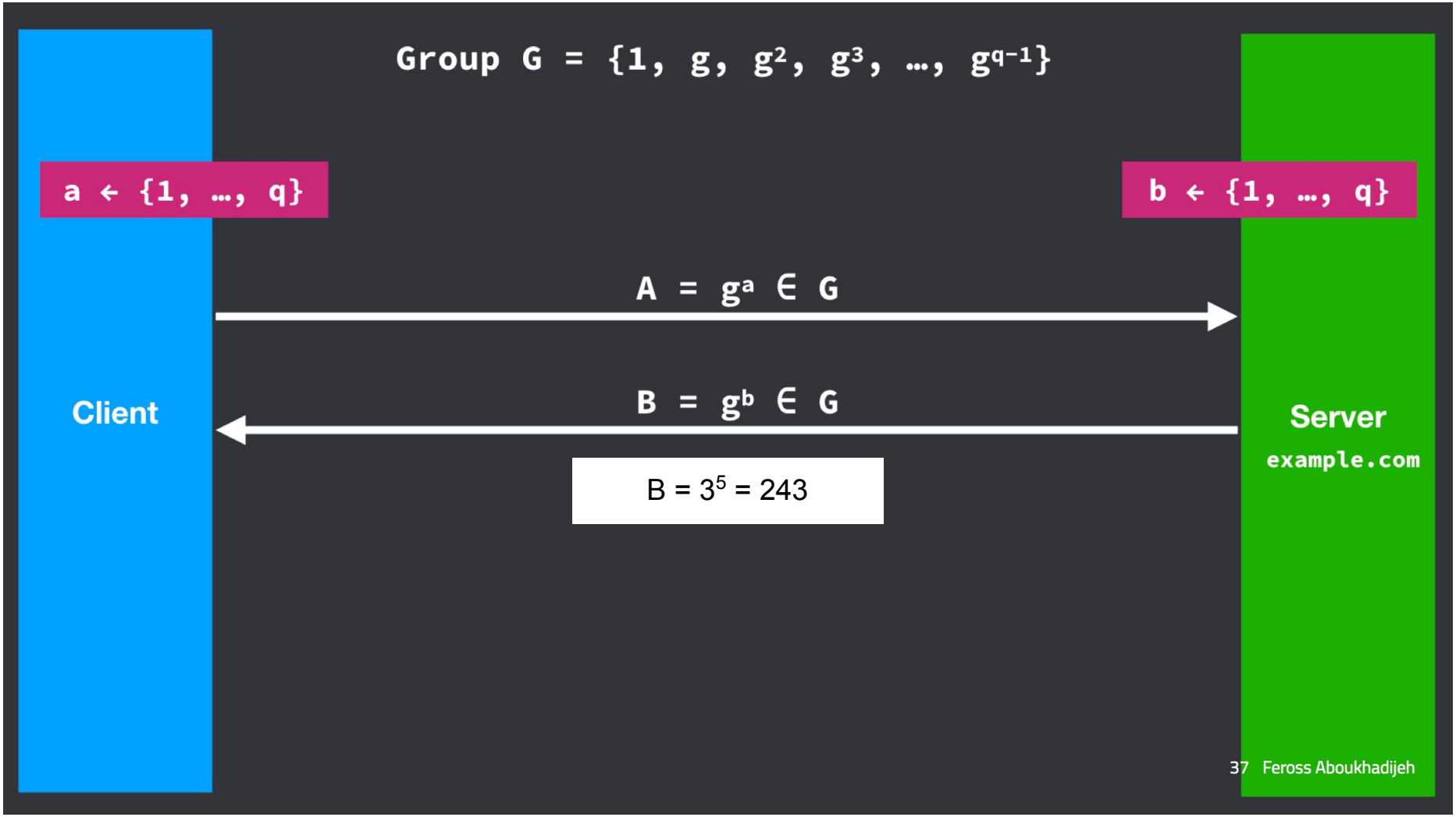

Client
This example will use small numbers for g, a, b, etc. but in practice these numbers will large enough to be computationally intimidating for factoring
Server
example.com
32 Feross Aboukhadijeh

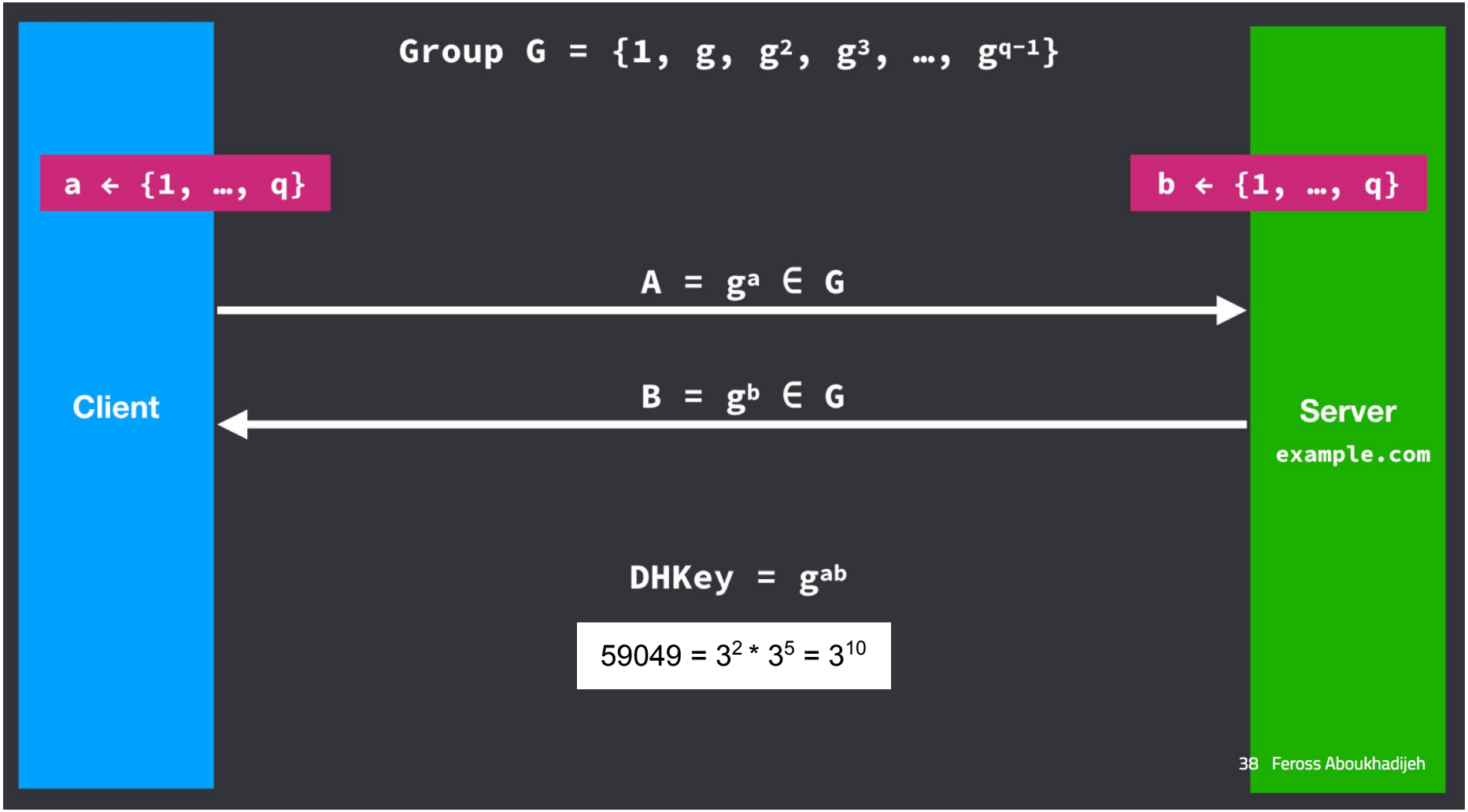

Group G = {1, g, g², g³, …, g^(q-1)}
g = 3, G = {3⁰, 3¹, 3², 3³, 3⁴, …, 3^(q-1)}
Client
Server
example.com
33 Feross Aboukhadijeh

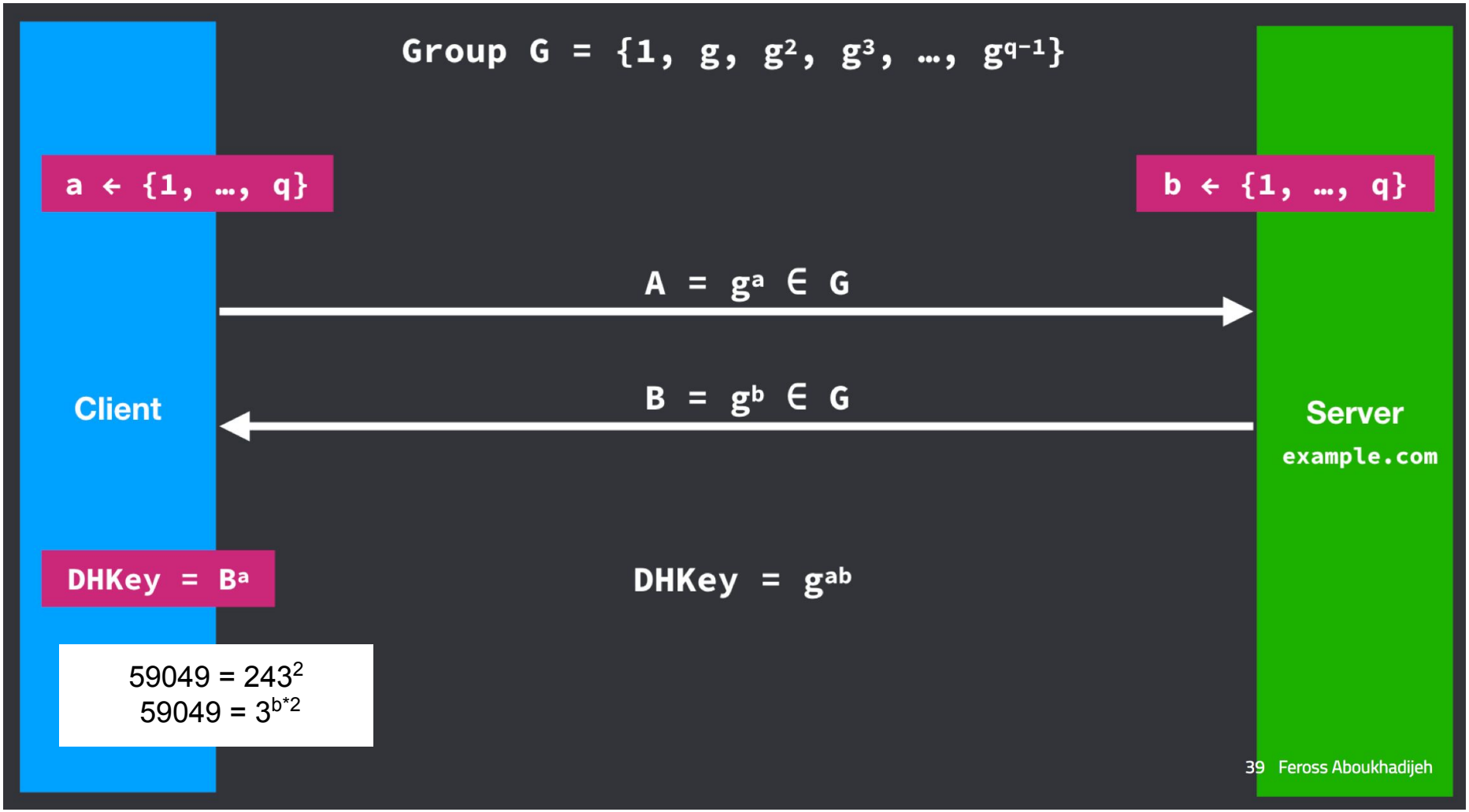

Group G = {1, g, g^2, g^3, …, g^(q-1)}
a ← {1, …, q}
a = 2
Client
Server
example.com
34 Feross Aboukhadijeh

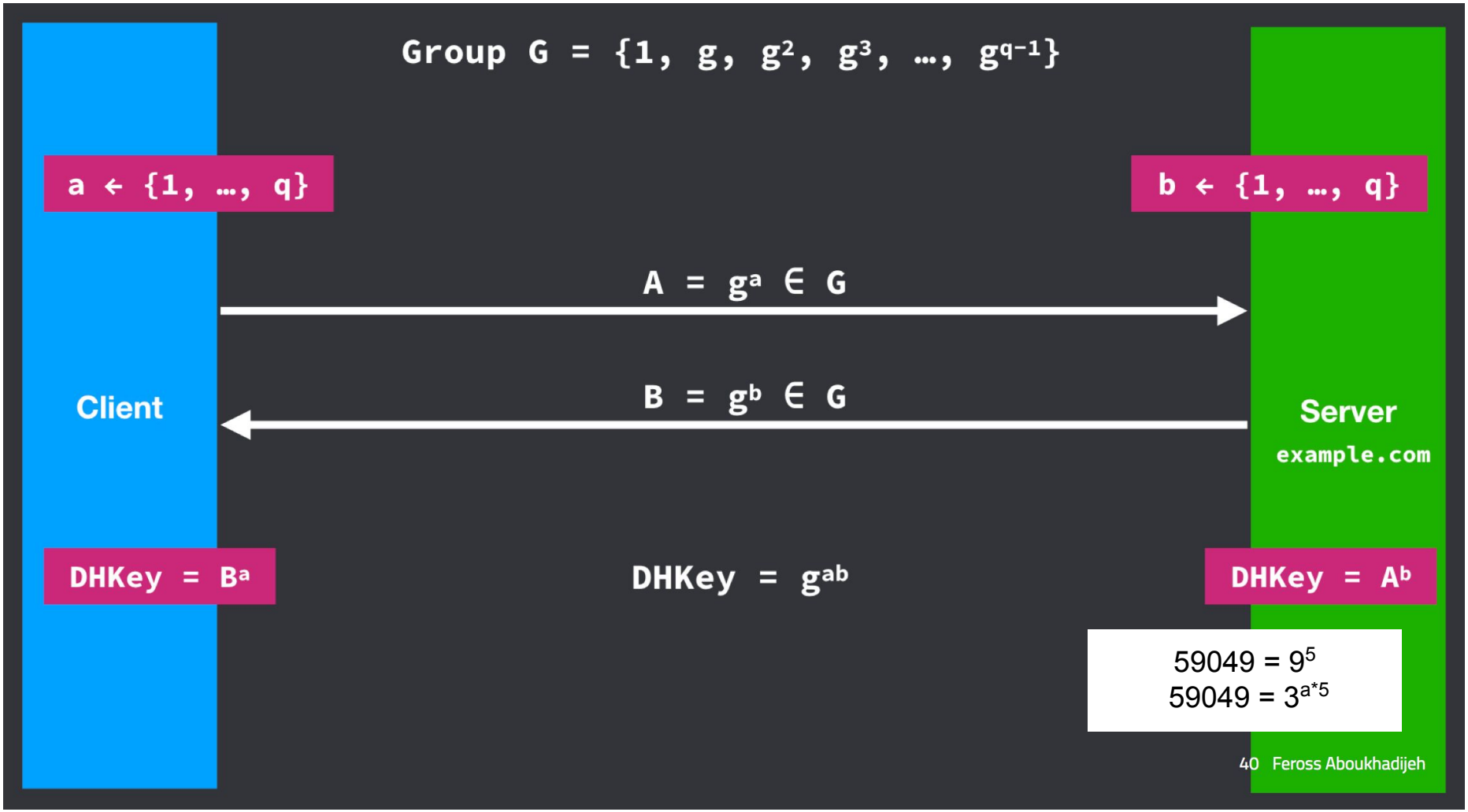

Group G = {1, g, g^2, g^3, …, g^(q-1)}
a ← {1, …, q}
b ← {1, …, q}
b = 5
Client
Server
example.com
35 Feross Aboukhadijeh

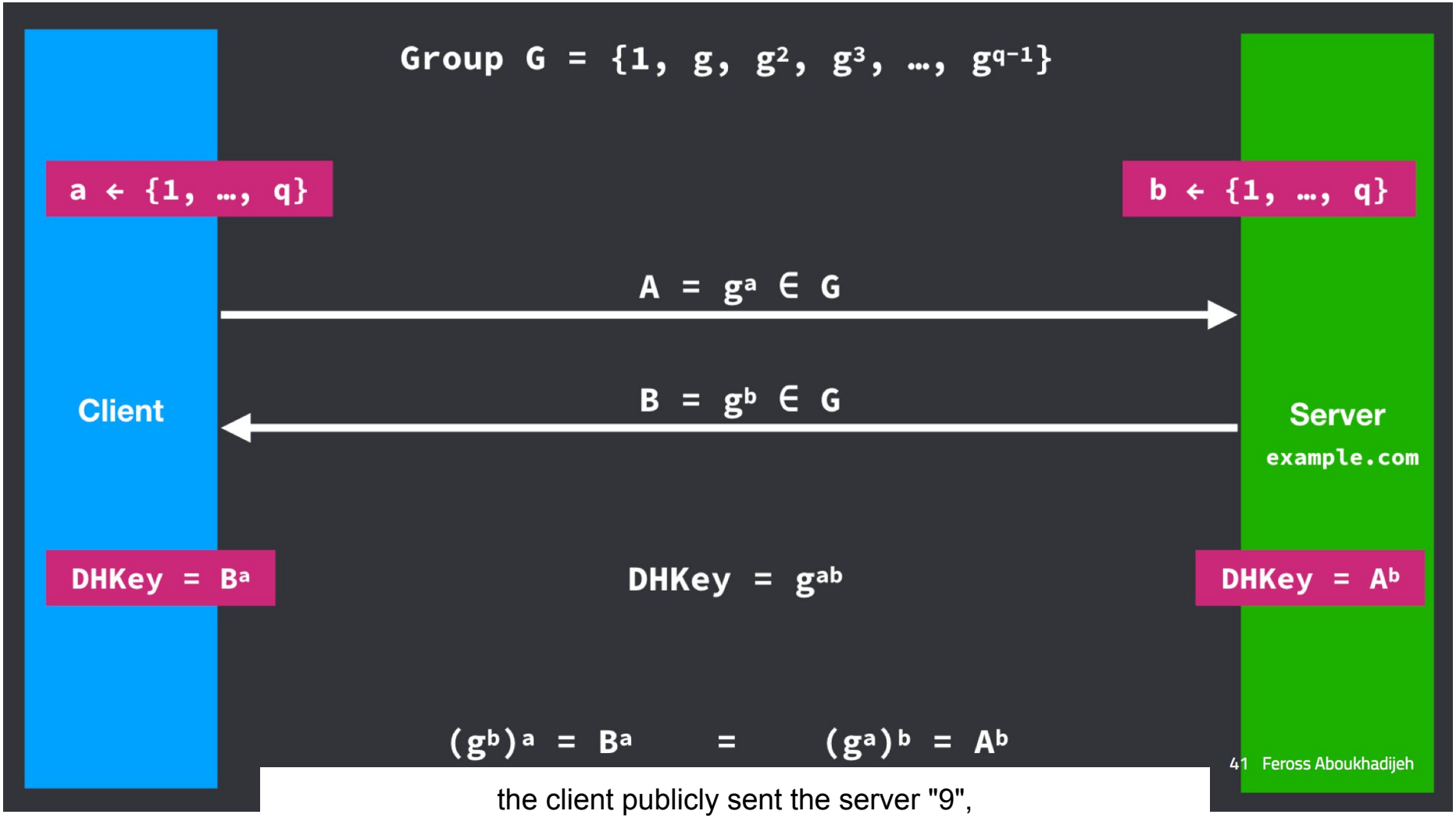

Group G = {1, g, g², g³, …, g^(q-1)}
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
A = 3² = 9
Client
Server
example.com
36 Feross Aboukhadijeh


Group G = {1, g, g^2, g^3, …, g^(q-1)}
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
B = g^b ∈ G
Client
Server
example.com
B = 3^5 = 243
37 Feross Aboukhadijeh


Group G = {1, g, g^2, g^3, …, g^(q-1)}
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
B = g^b ∈ G
Client
Server
example.com
DHKey = g^ab
59049 = 3^2 * 3^5 = 3^10
38 Feross Aboukhadijeh

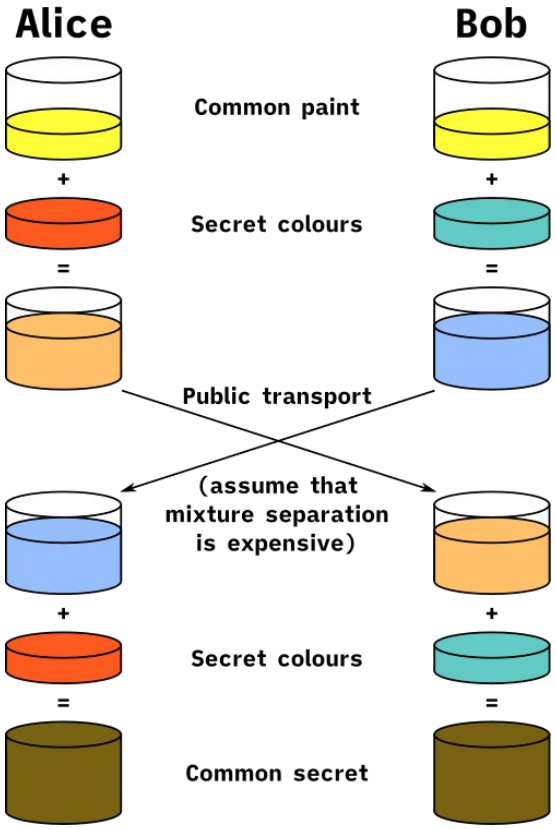

Group G = {1, g, g², g³, …, g^(q-1)}
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
B = g^b ∈ G
Client
Server
example.com
DHKey = B^a
DHKey = g^ab
59049 = 243²
59049 = 3^(b*2)
39 Feross Aboukhadijeh


Group G = {1, g, g^2, g^3, …, g^(q-1)}
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
B = g^b ∈ G
Client
Server
example.com
DHKey = B^a
DHKey = g^ab
DHKey = A^b
59049 = 9^5
59049 = 3^(a*5)
40 Feross Aboukhadijeh




the client publicly sent the server "9",
the server publicly sent the client "243",
and now they've both agreed to use "59049"
anyone listening who heard "9" & "243" can't easily arrive at "59049"

# Two more (simplified) explanations




**Cryptographic explanation** [ edit ]

The simplest and the original implementation[2] of the protocol uses the multiplicative group of integers modulo $p$, where $p$ is prime, and $g$ is a primitive root modulo $p$. These two values are chosen in this way to ensure that the resulting shared secret can take on any value from 1 to $p$–1. Here is an example of the protocol, with non-secret values in blue, and secret values in **red**.

1. Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
2. Alice chooses a secret integer $\mathbf{a} = 4$, then sends Bob $A = g^{\mathbf{a}} \bmod p$
   - $A = 5^4 \bmod 23 = 4$
3. Bob chooses a secret integer $\mathbf{b} = 3$, then sends Alice $B = g^{\mathbf{b}} \bmod p$
   - $B = 5^3 \bmod 23 = 10$
4. Alice computes $\mathbf{s} = B^{\mathbf{a}} \bmod p$
   - $\mathbf{s} = 10^4 \bmod 23 = 18$
5. Bob computes $\mathbf{s} = A^{\mathbf{b}} \bmod p$
   - $\mathbf{s} = 4^3 \bmod 23 = 18$
6. Alice and Bob now share a secret (the number 18).

Both Alice and Bob have arrived at the same values because under mod p,

$$A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$$

https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

# Anonymous key exchange

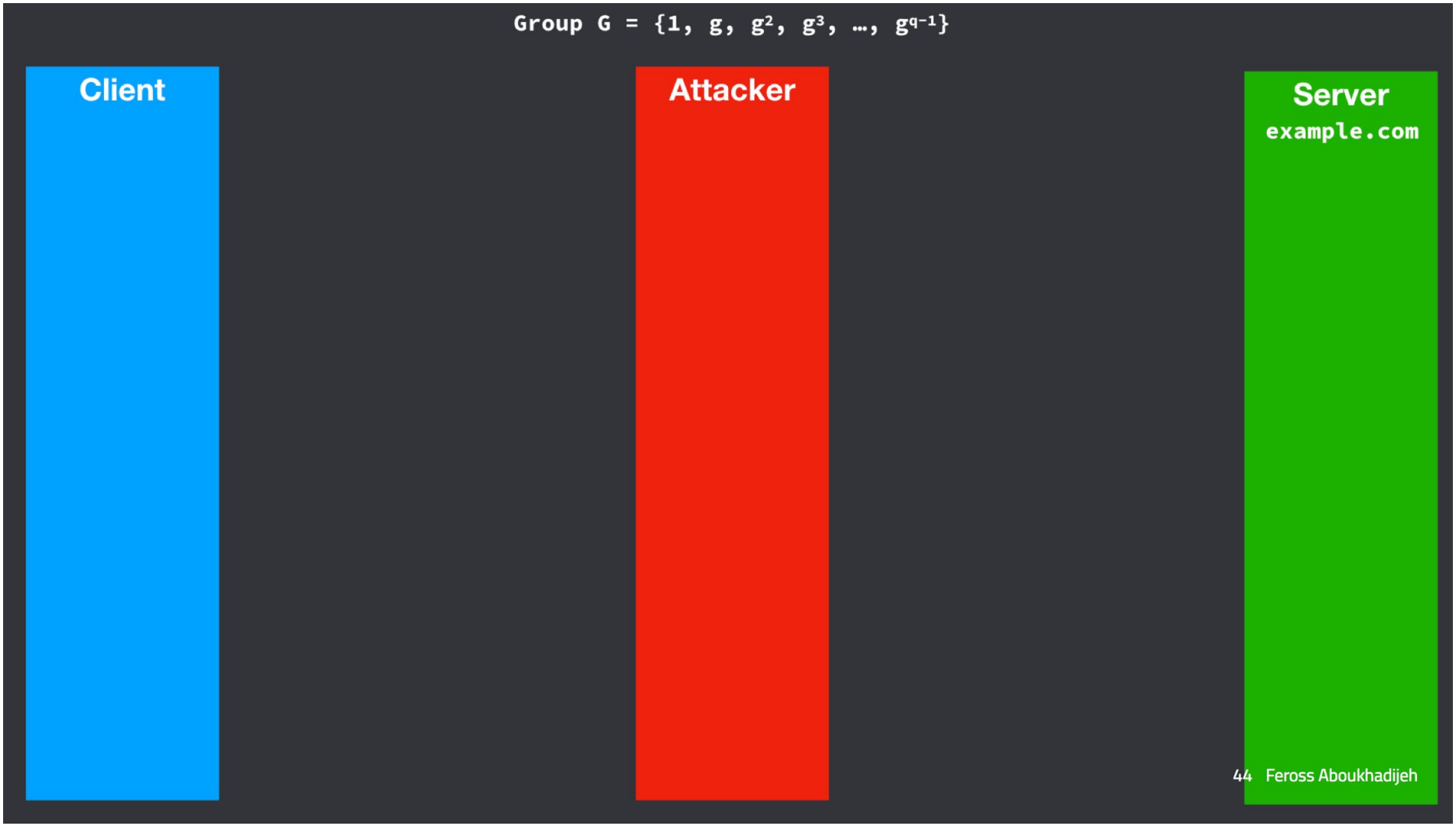
- Problem: Client doesn't know with which server it performed key exchange
  - It's possible that the client securely derived a key with the network attacker instead of the intended server!
- While the communication is technically private (secure against eavesdropping), it lacks authentication
  - Key idea: Without authentication, you can't actually have privacy

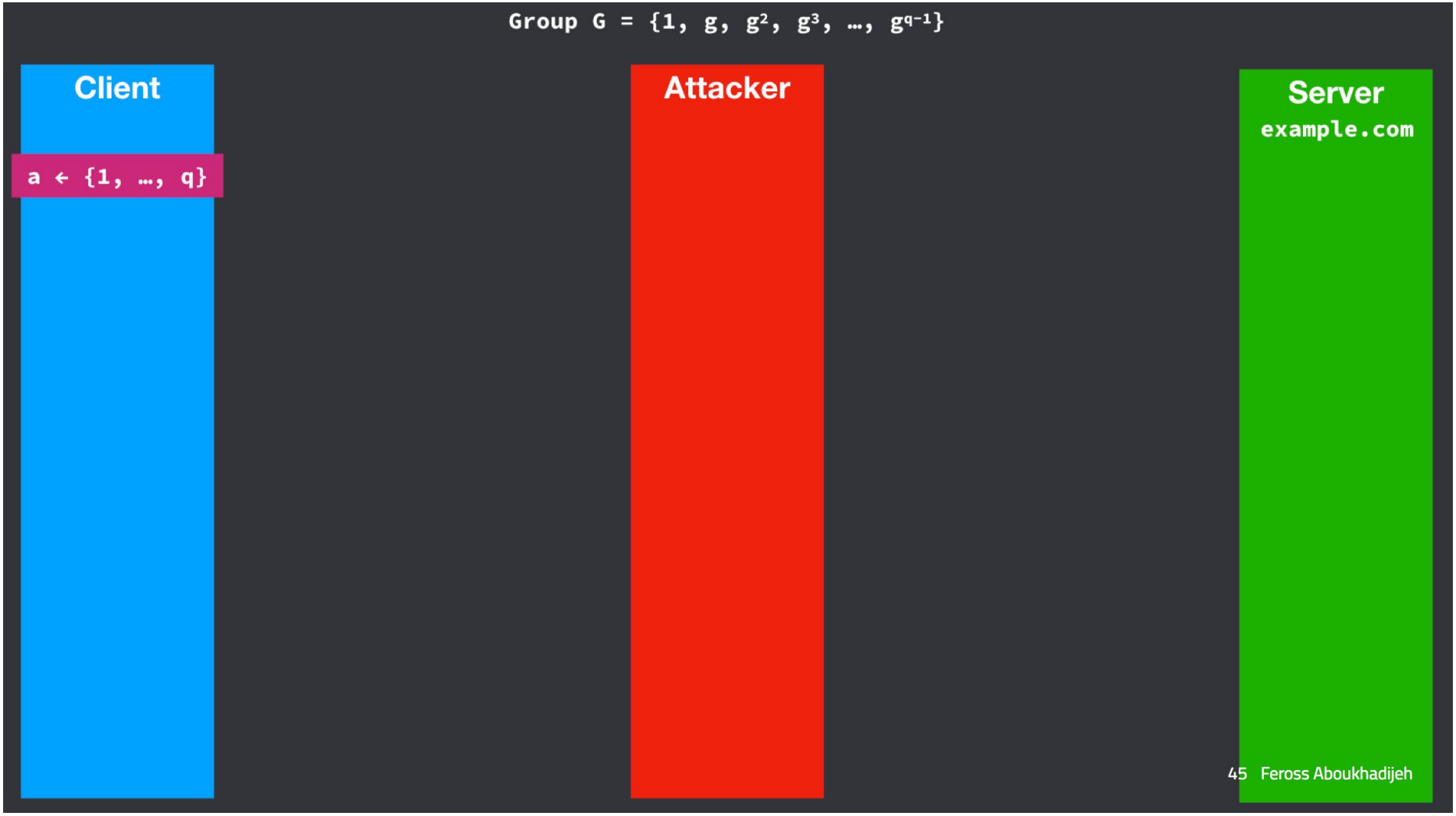
Group G = {1, g, $g^2$, $g^3$, …, $g^{q-1}$}

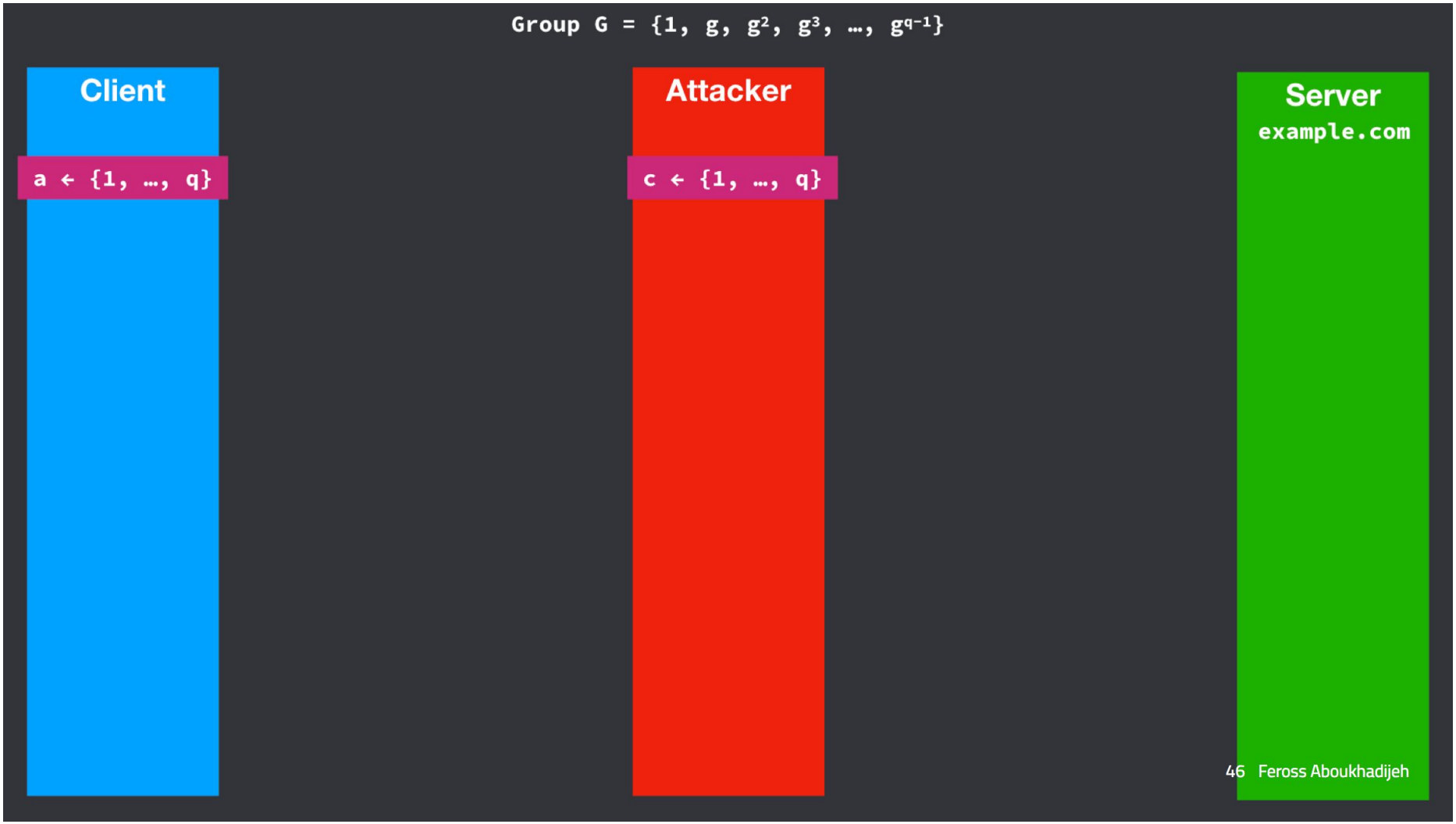
# Man-in-the-middle attack on Anonymous Diffie-Helman key exchange

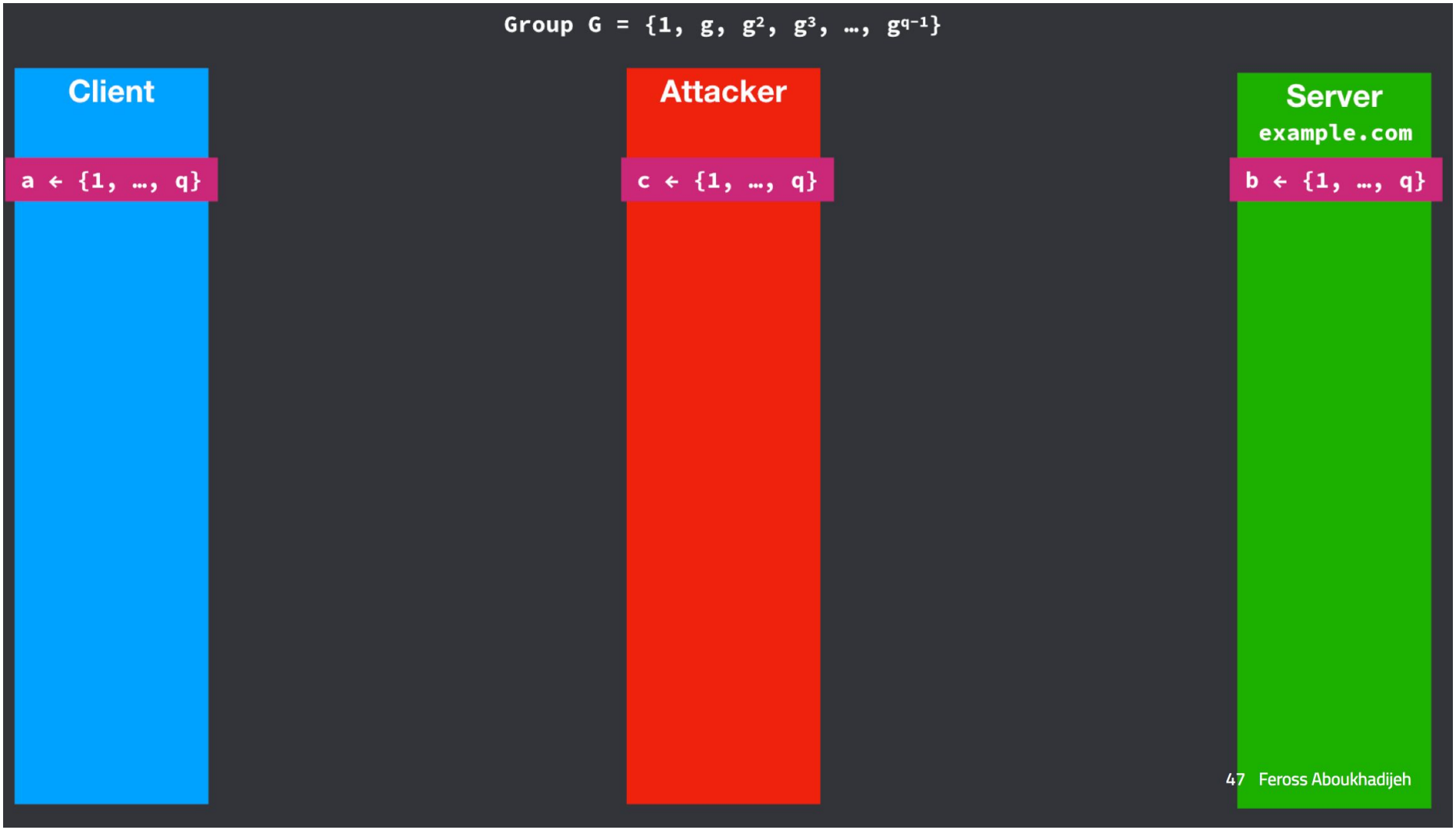
Server
example.com

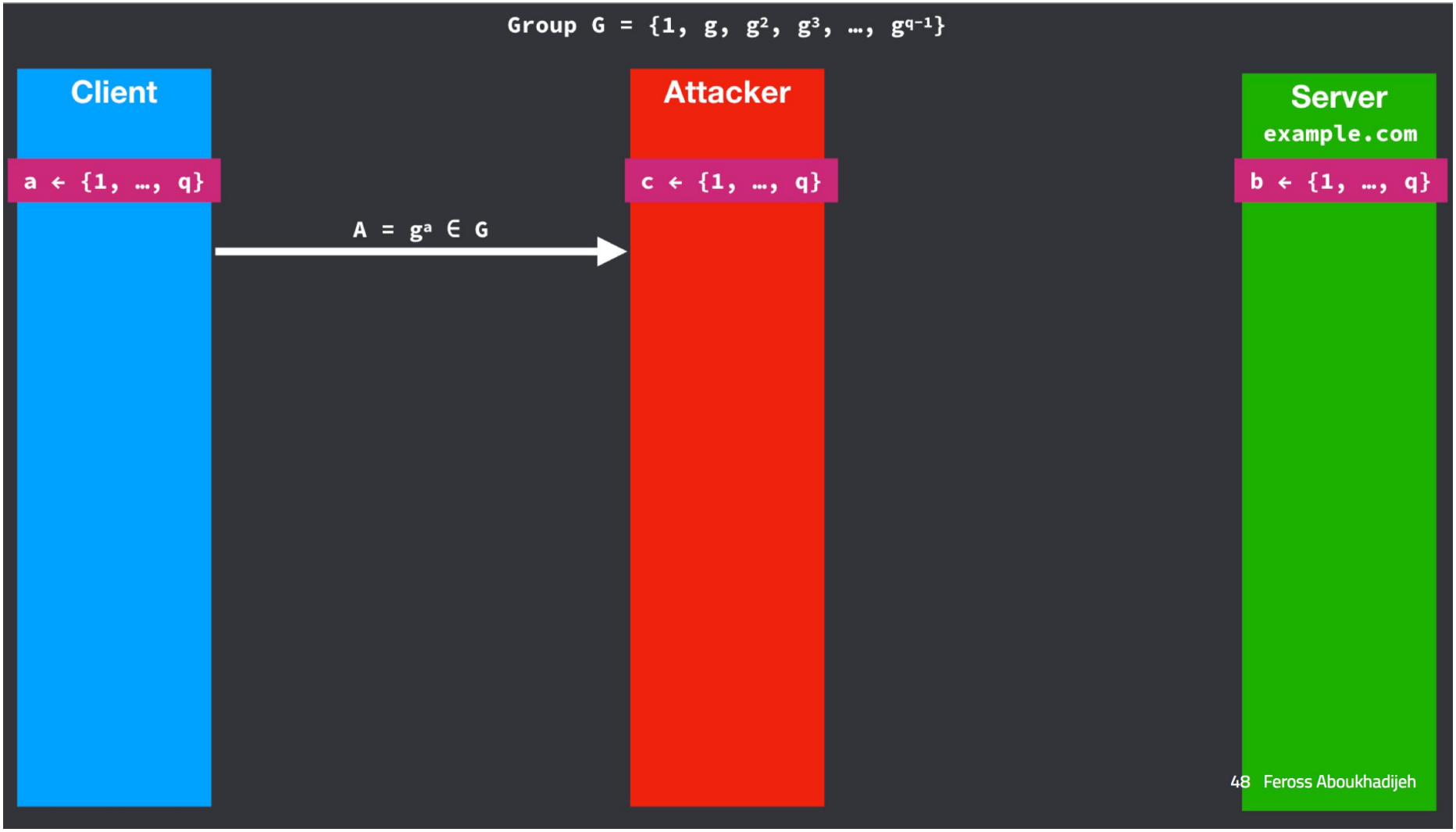
43 Feross Aboukhadijeh

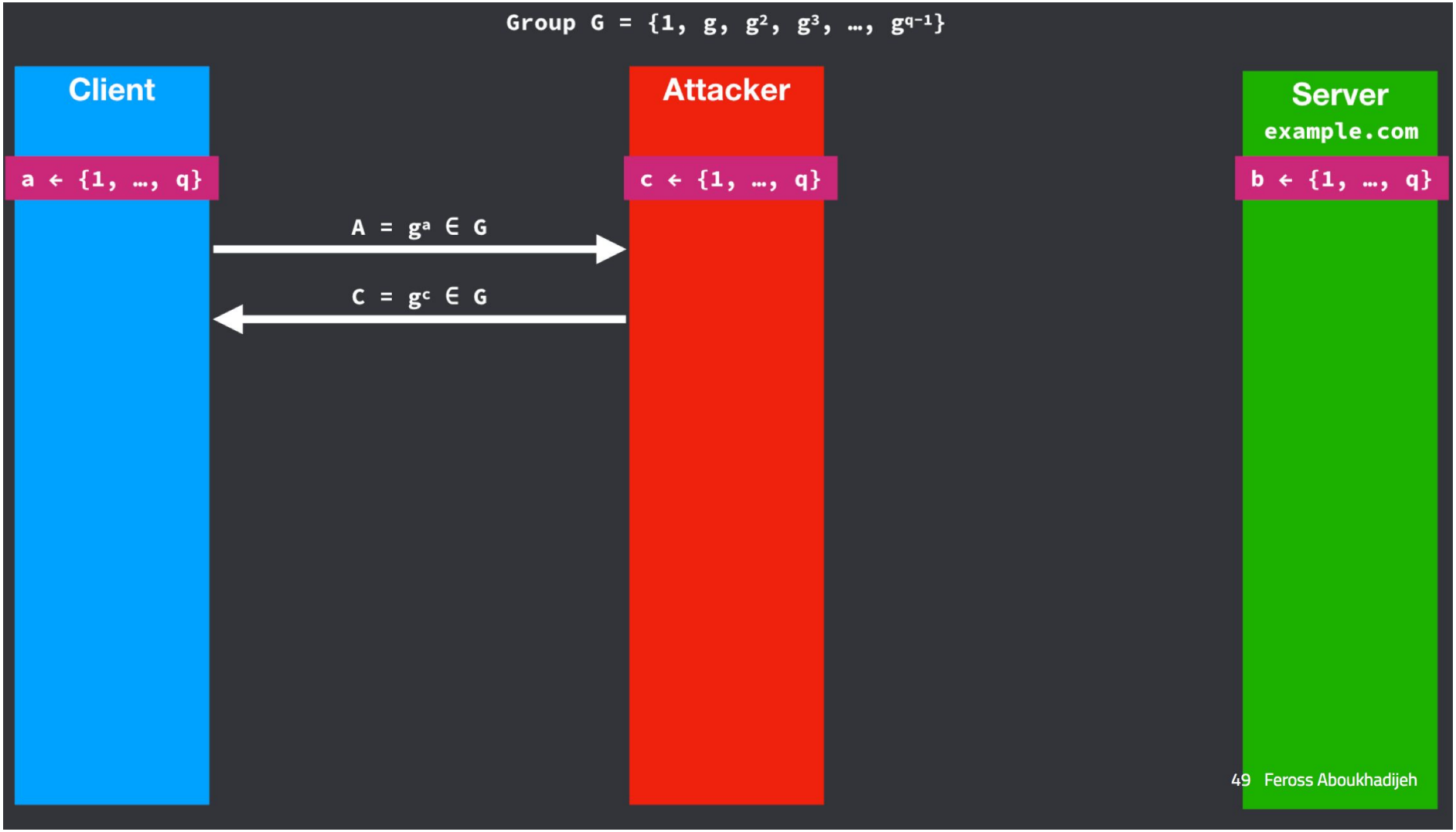

Group G = {1, g, g², g³, …, g^(q-1)}
Client
Attacker
Server
example.com
44 Feross Aboukhadijeh

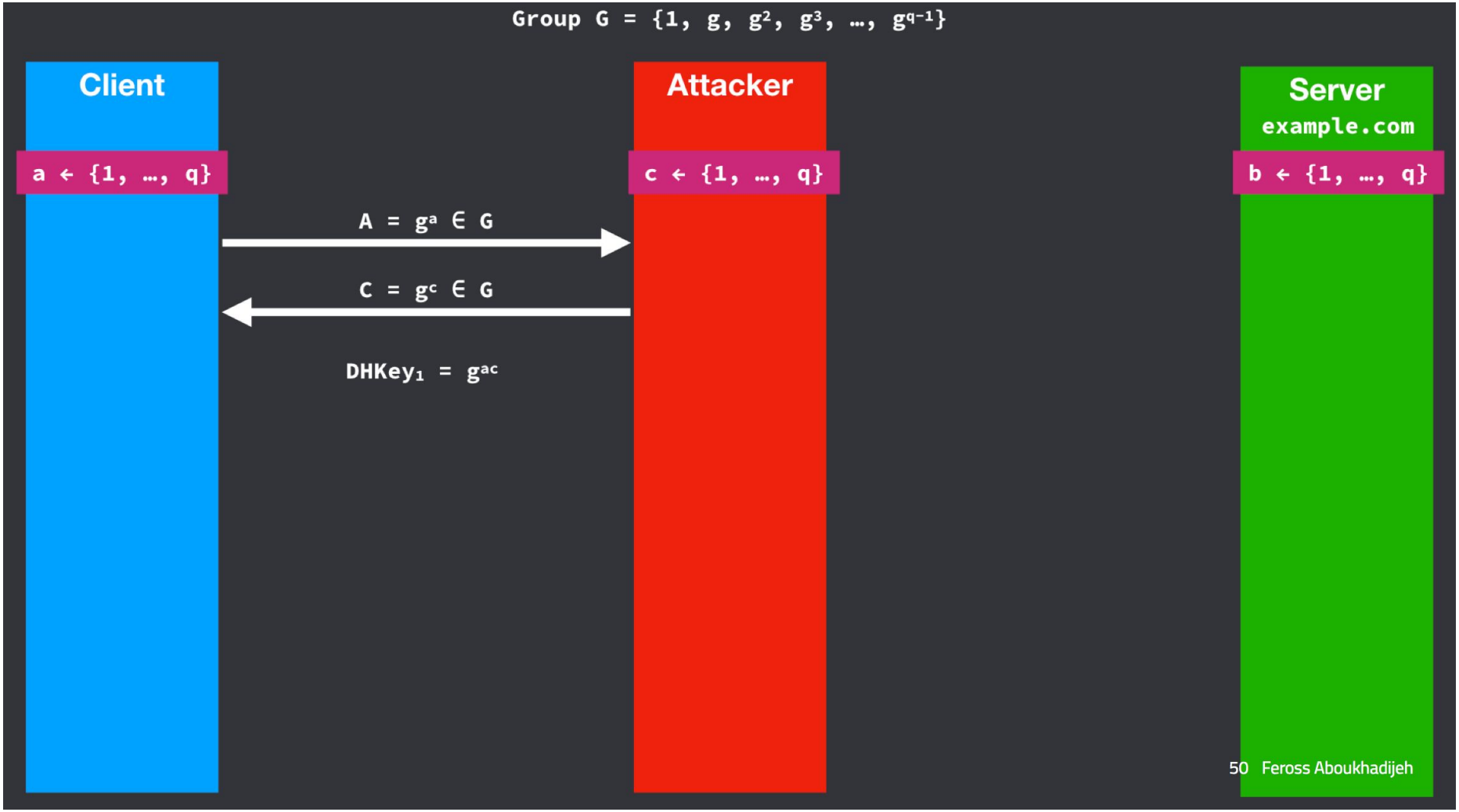

Group G = {1, g, g², g³, …, g^(q-1)}
Client
a ← {1, …, q}
Attacker
Server
example.com
45 Feross Aboukhadijeh

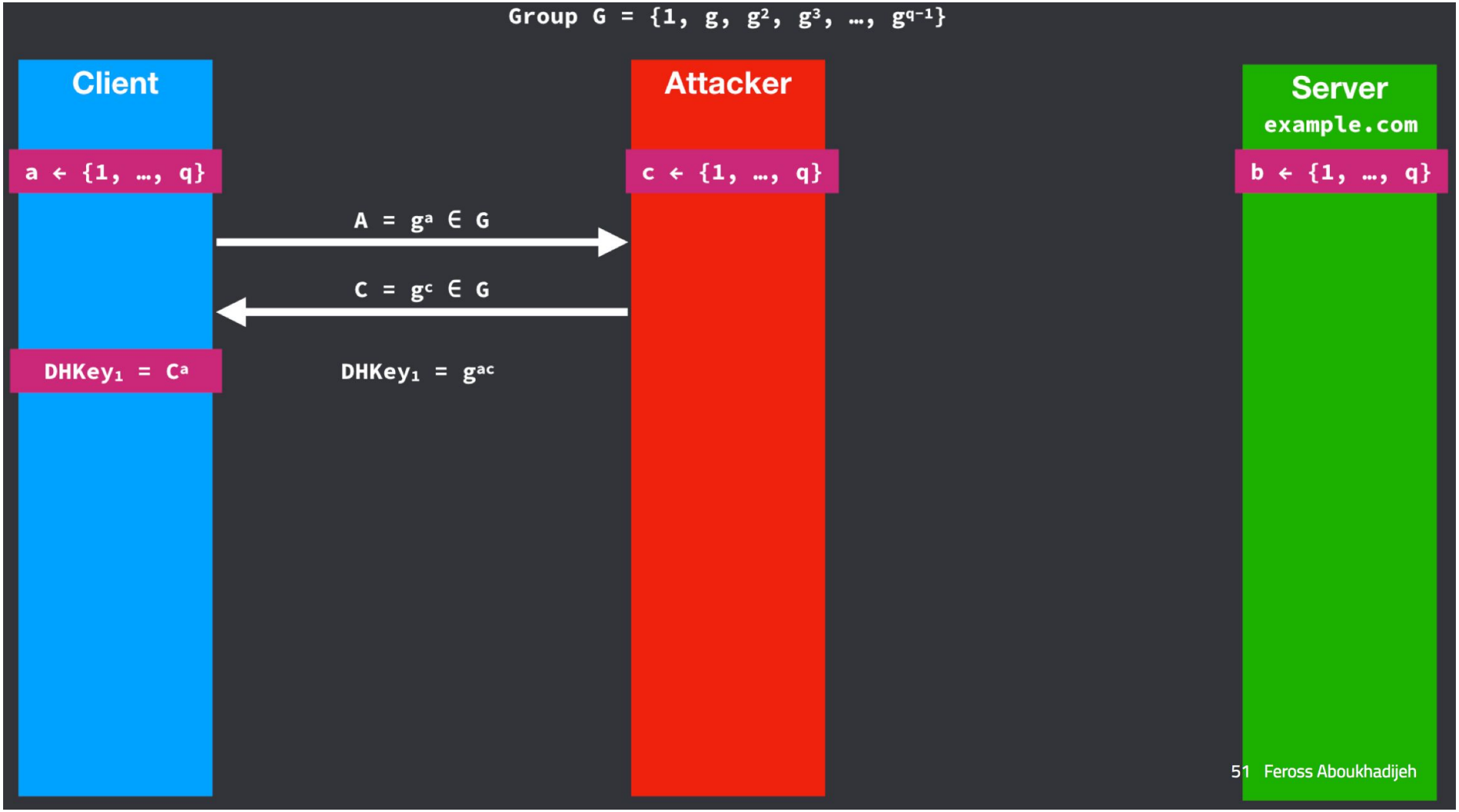

Group G = {1, g, g², g³, …, g^(q−1)}
Client
a ← {1, …, q}
Attacker
c ← {1, …, q}
Server
example.com
46 Feross Aboukhadijeh

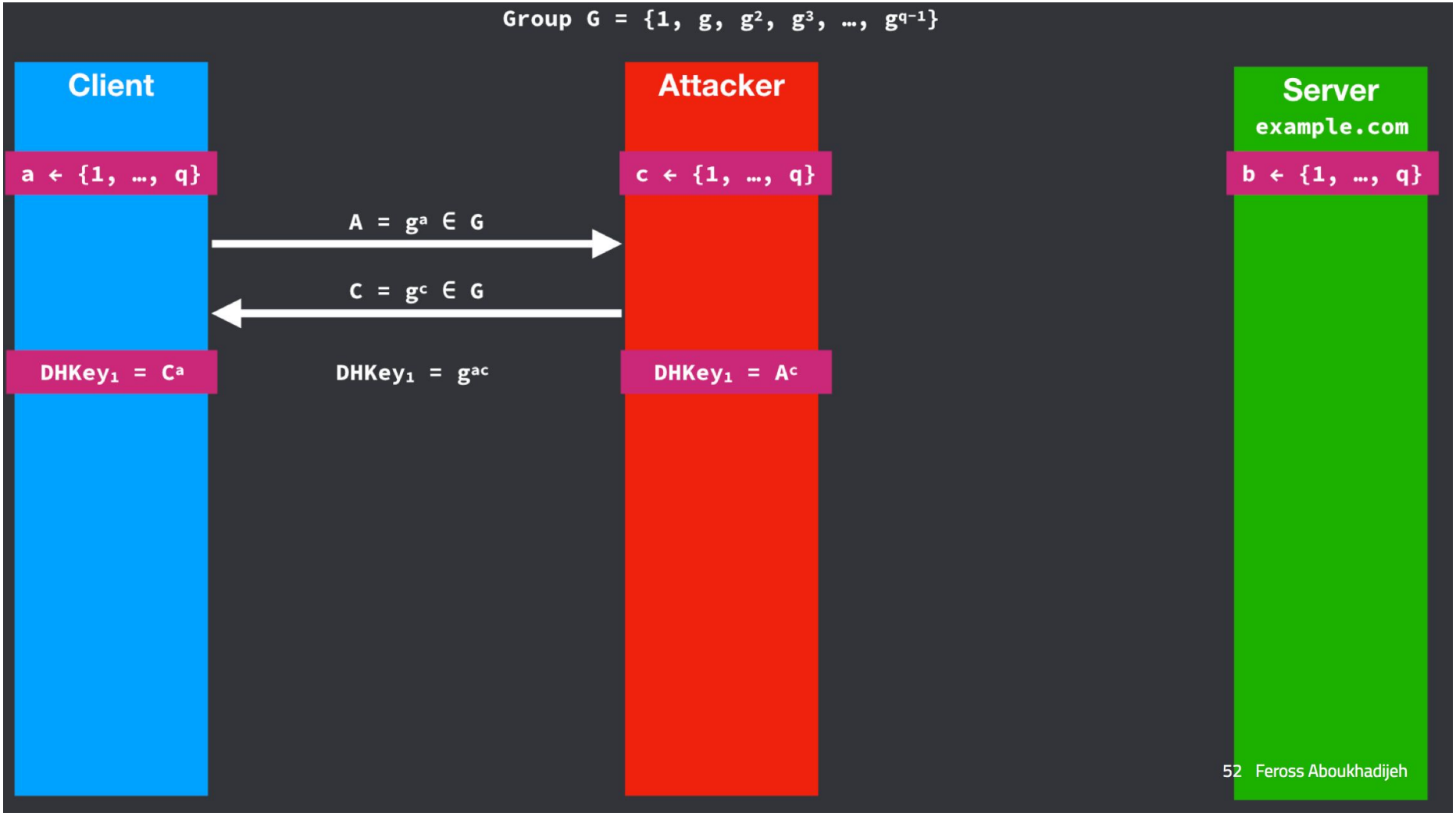

Group G = {1, g, g², g³, …, g^(q−1)}
Client
a ← {1, …, q}
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
47 Feross Aboukhadijeh

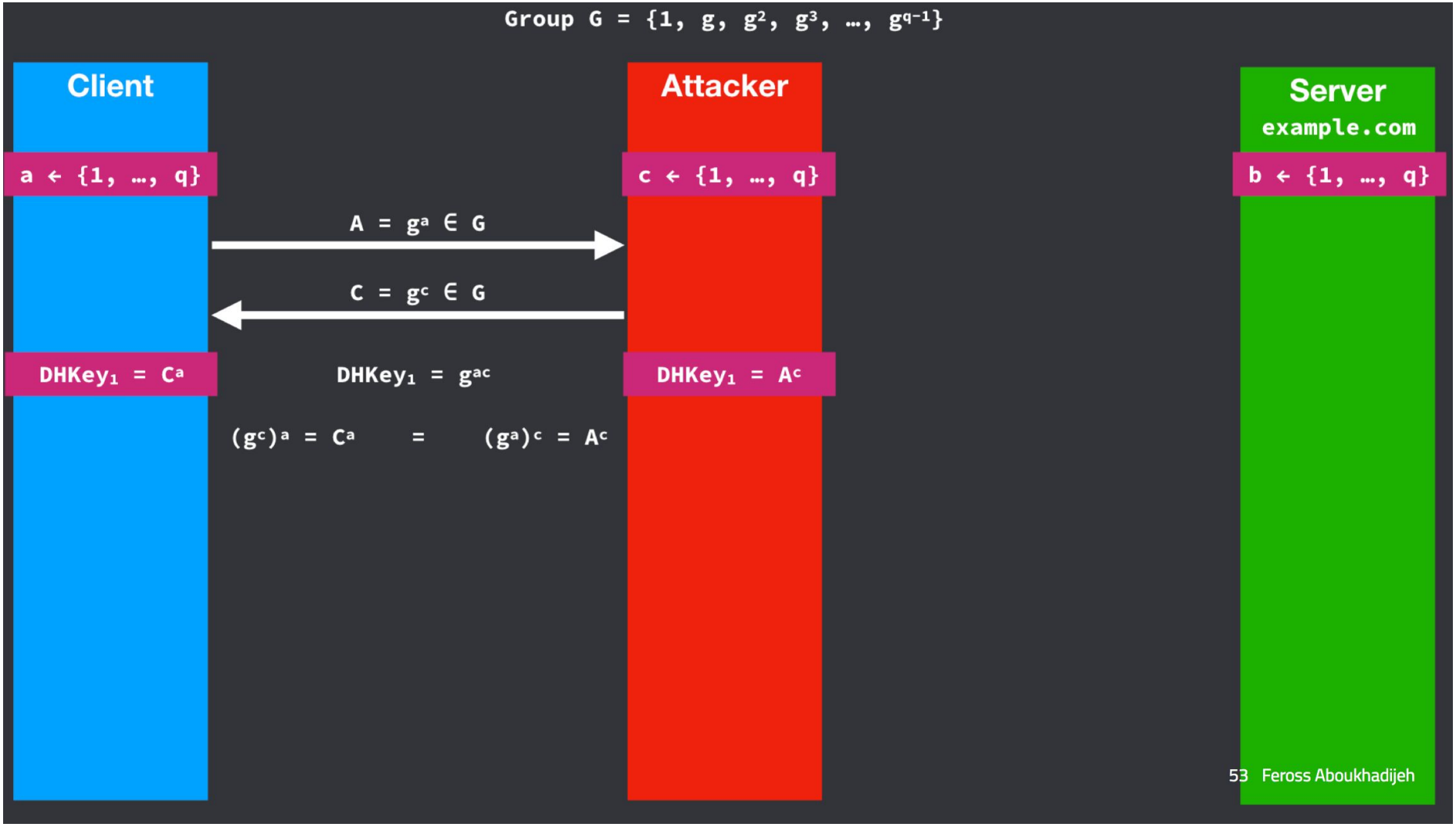

Group G = {1, g, g², g³, …, g^(q-1)}
Client
a ← {1, …, q}
A = g^a ∈ G
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
48 Feross Aboukhadijeh

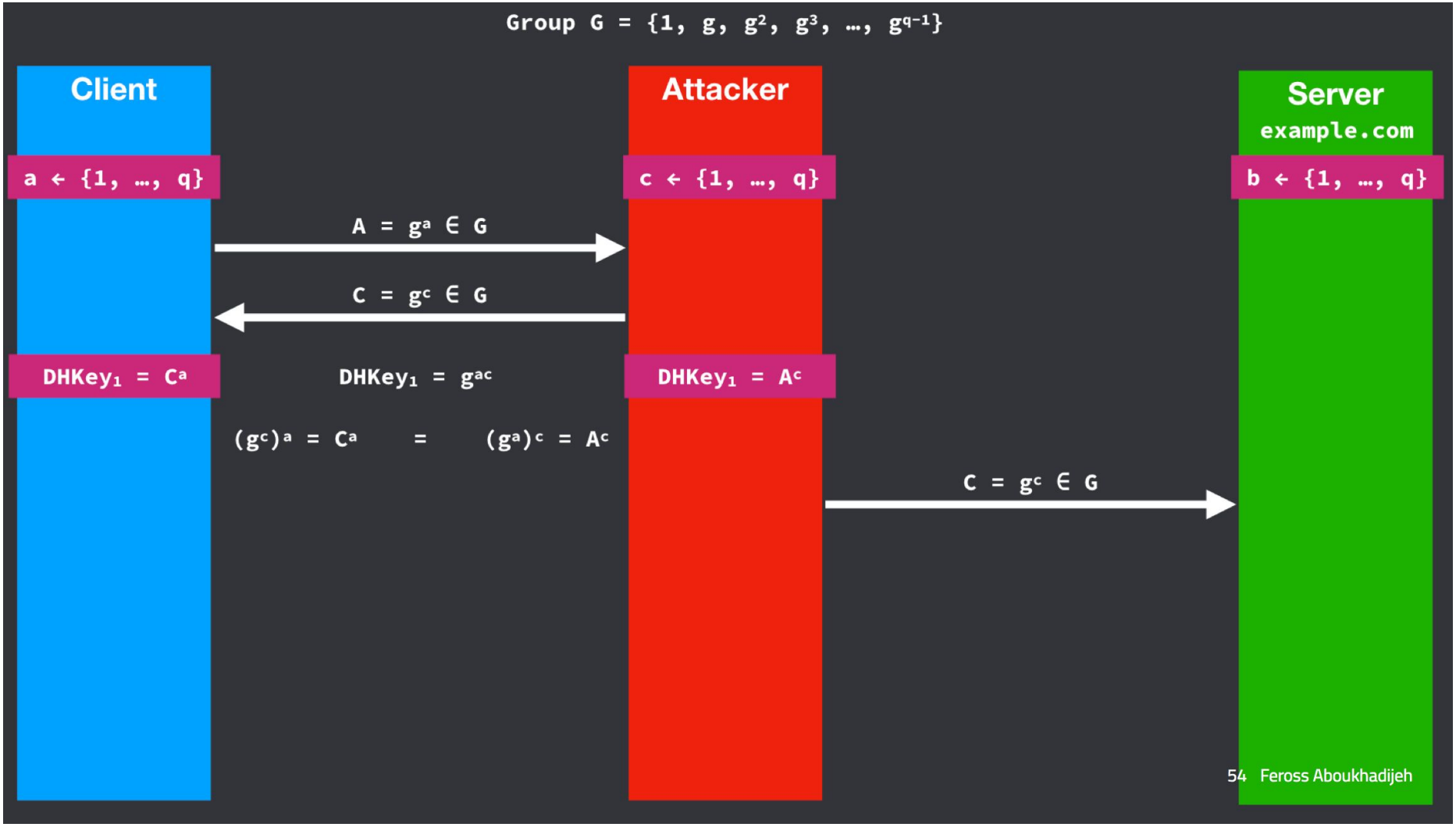

Group G = {1, g, g², g³, …, g^(q-1)}
Client
a ← {1, …, q}
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
49 Feross Aboukhadijeh

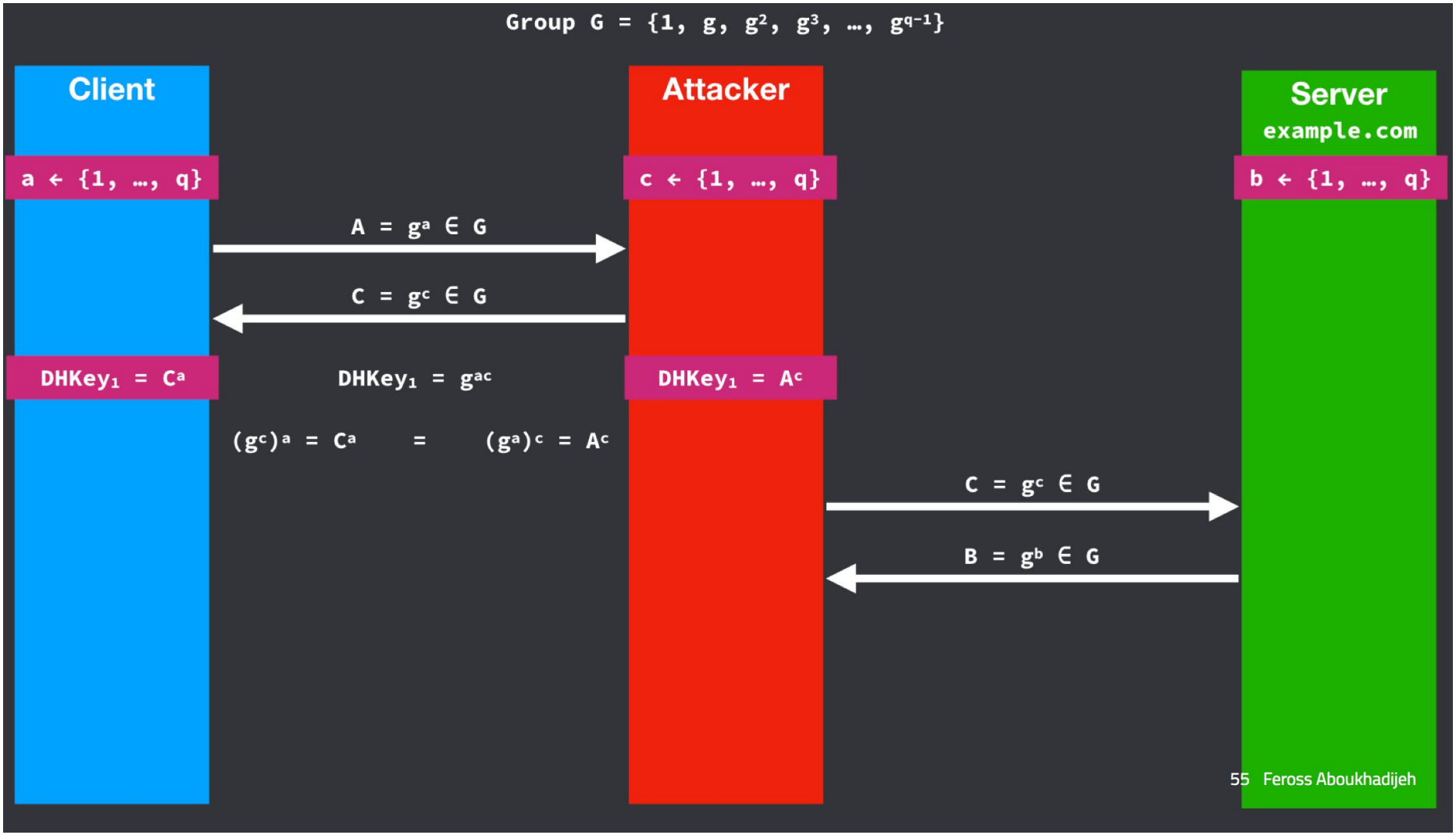

Group G = {1, g, g², g³, …, g^(q-1)}
Client
a ← {1, …, q}
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = g^ac
50 Feross Aboukhadijeh

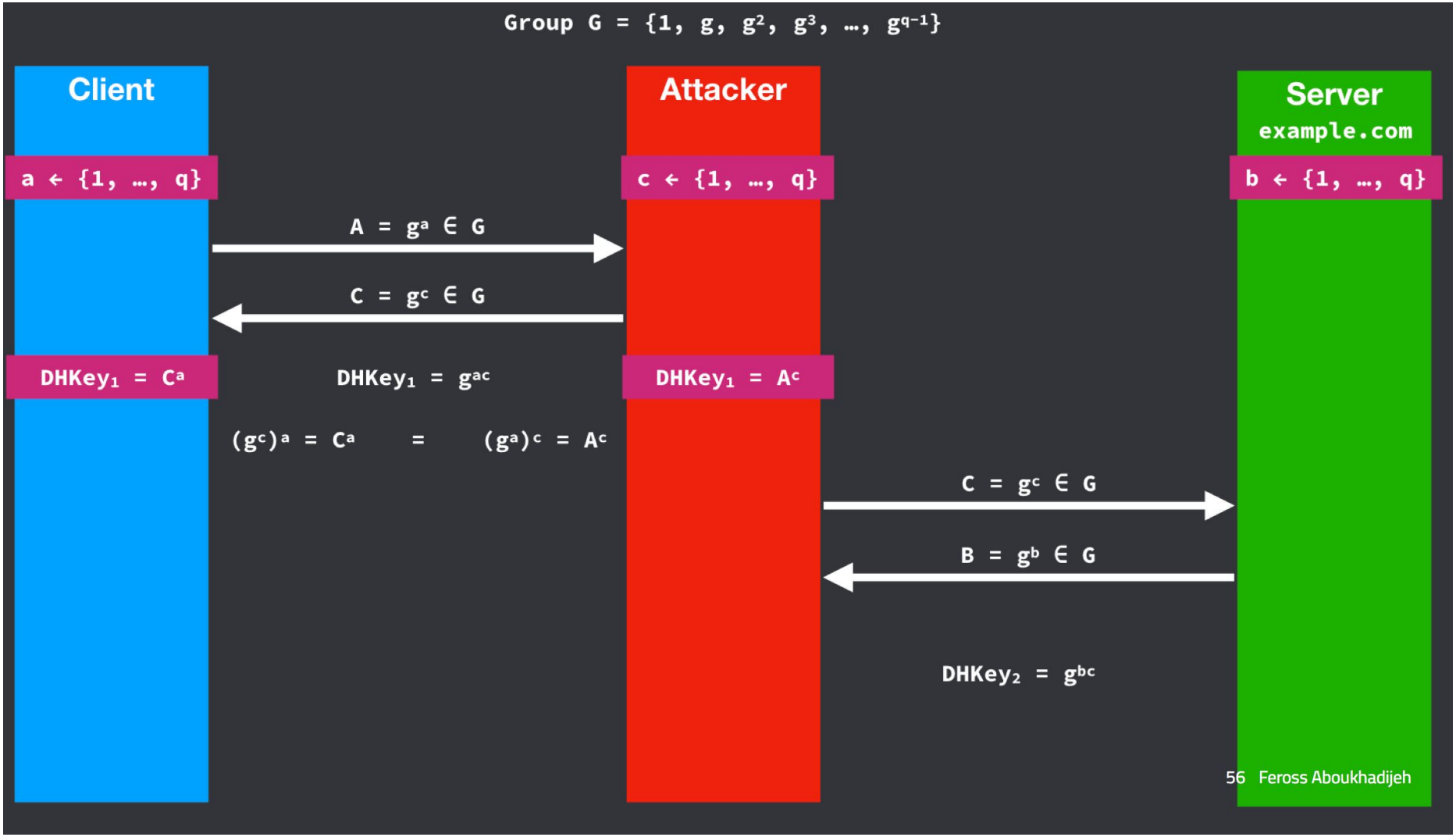

Group G = {1, g, g², g³, …, g^(q-1)}
Client
a ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey₁ = C^a
DHKey₁ = g^ac
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
51 Feross Aboukhadijeh

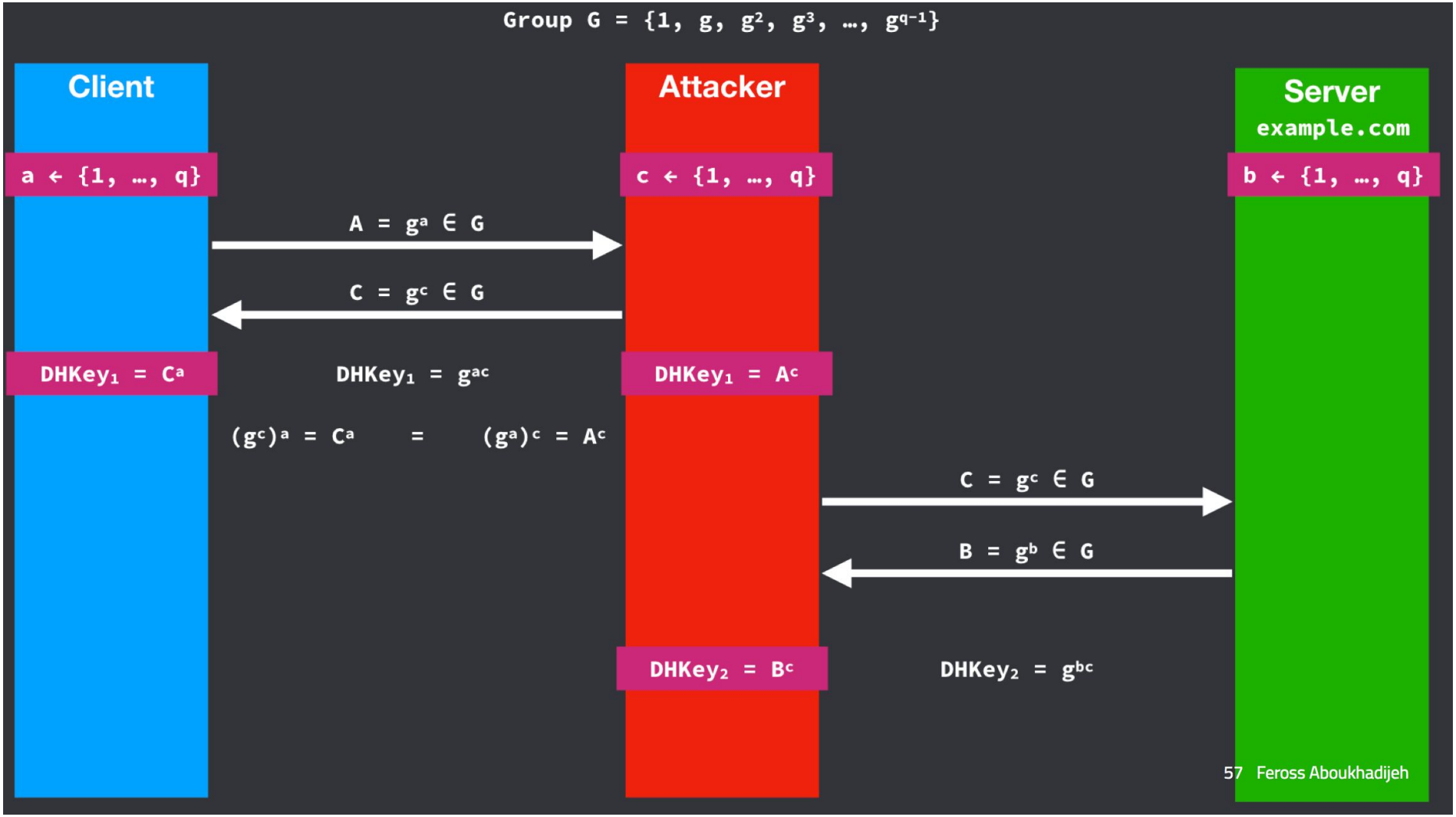

Group G = {1, g, g², g³, …, g^(q-1)}
Client
Attacker
Server
example.com
a ← {1, …, q}
c ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
52 Feross Aboukhadijeh

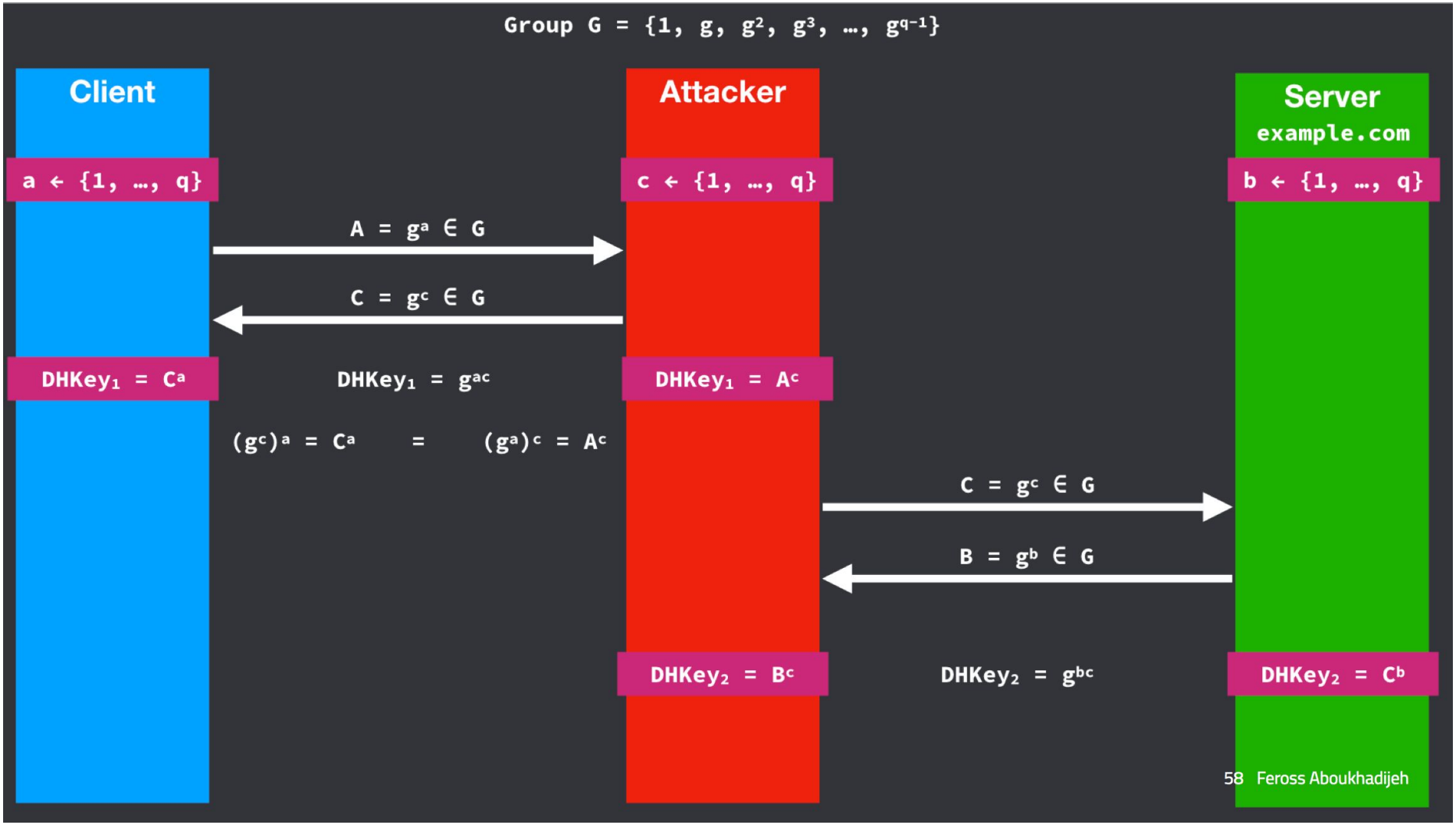

Group G = {1, g, g², g³, …, g^(q−1)}
Client
a ← {1, …, q}
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
53 Feross Aboukhadijeh

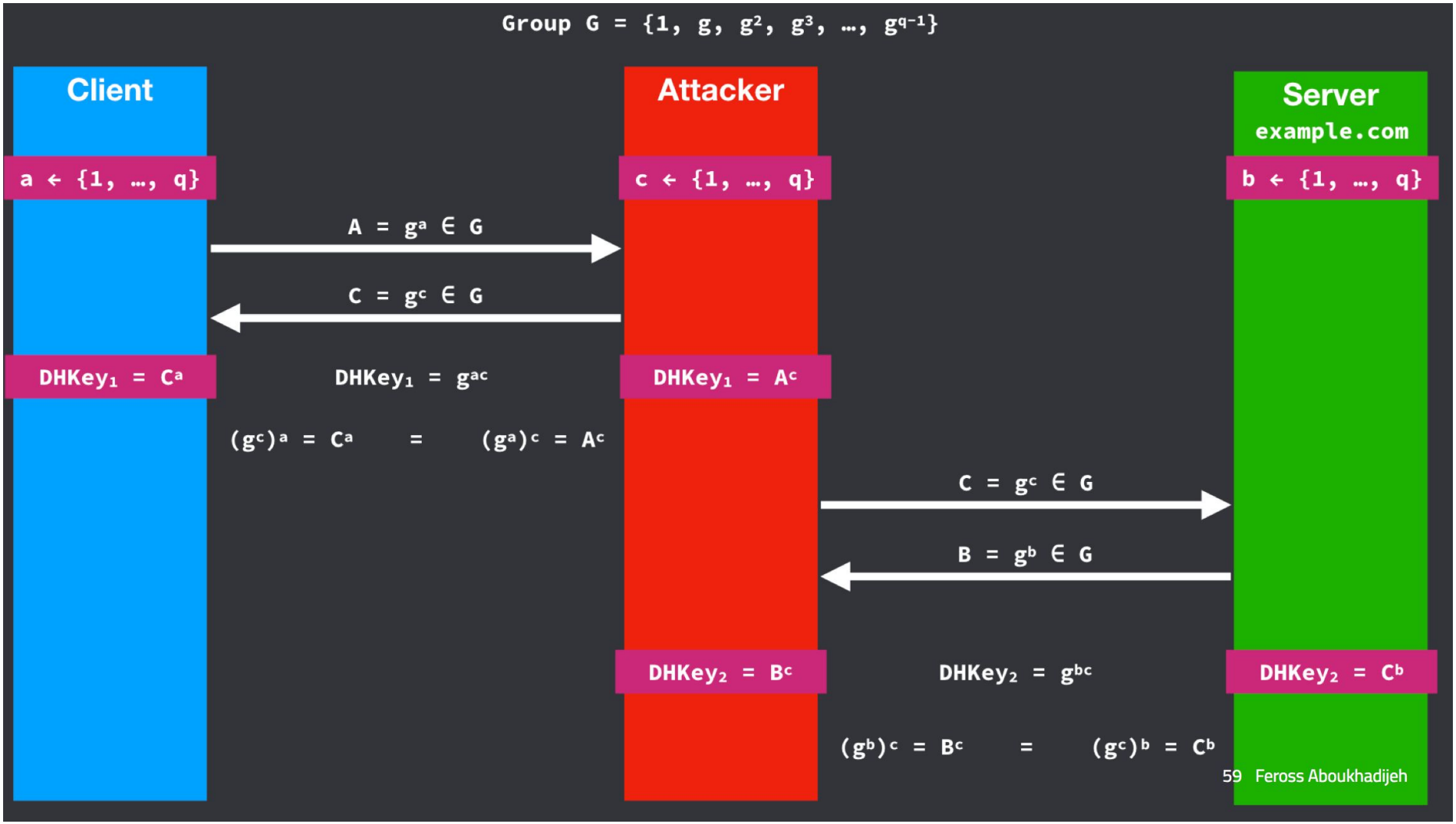

Group G = {1, g, g², g³, …, g^(q-1)}
Client
Attacker
Server
example.com
a ← {1, …, q}
c ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
C = g^c ∈ G
54 Feross Aboukhadijeh


Group G = {1, g, g², g³, …, g^(q-1)}
Client
a ← {1, …, q}
Attacker
c ← {1, …, q}
Server
example.com
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
C = g^c ∈ G
B = g^b ∈ G
55 Feross Aboukhadijeh


Group G = {1, g, g², g³, …, g^(q−1)}
Client
Attacker
Server
example.com
a ← {1, …, q}
c ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
C = g^c ∈ G
B = g^b ∈ G
DHKey2 = g^bc
56 Feross Aboukhadijeh


Group G = {1, g, g², g³, …, g^(q-1)}
Client
Attacker
Server
example.com
a ← {1, …, q}
c ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
C = g^c ∈ G
B = g^b ∈ G
DHKey2 = B^c
DHKey2 = g^bc
57 Feross Aboukhadijeh


Group G = {1, g, g², g³, …, g^(q-1)}
Client
Attacker
Server
example.com
a ← {1, …, q}
c ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
C = g^c ∈ G
B = g^b ∈ G
DHKey2 = B^c
DHKey2 = g^bc
DHKey2 = C^b
58 Feross Aboukhadijeh


Group G = {1, g, g², g³, …, g^(q-1)}
Client
Attacker
Server
example.com
a ← {1, …, q}
c ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
C = g^c ∈ G
DHKey1 = C^a
DHKey1 = g^ac
DHKey1 = A^c
(g^c)^a = C^a = (g^a)^c = A^c
C = g^c ∈ G
B = g^b ∈ G
DHKey2 = B^c
DHKey2 = g^bc
DHKey2 = C^b
(g^b)^c = B^c = (g^c)^b = C^b
59 Feross Aboukhadijeh

# How do we get authentication?

- Goal: If the client could authenticate the server it is performing key exchange with, then it could securely derive a shared key with that (and only that) server
- Solution: Use public-key cryptography for authentication
  - Remember signing from lecture 2 on cookies?
  - Let's review

# Review: Signature schemes

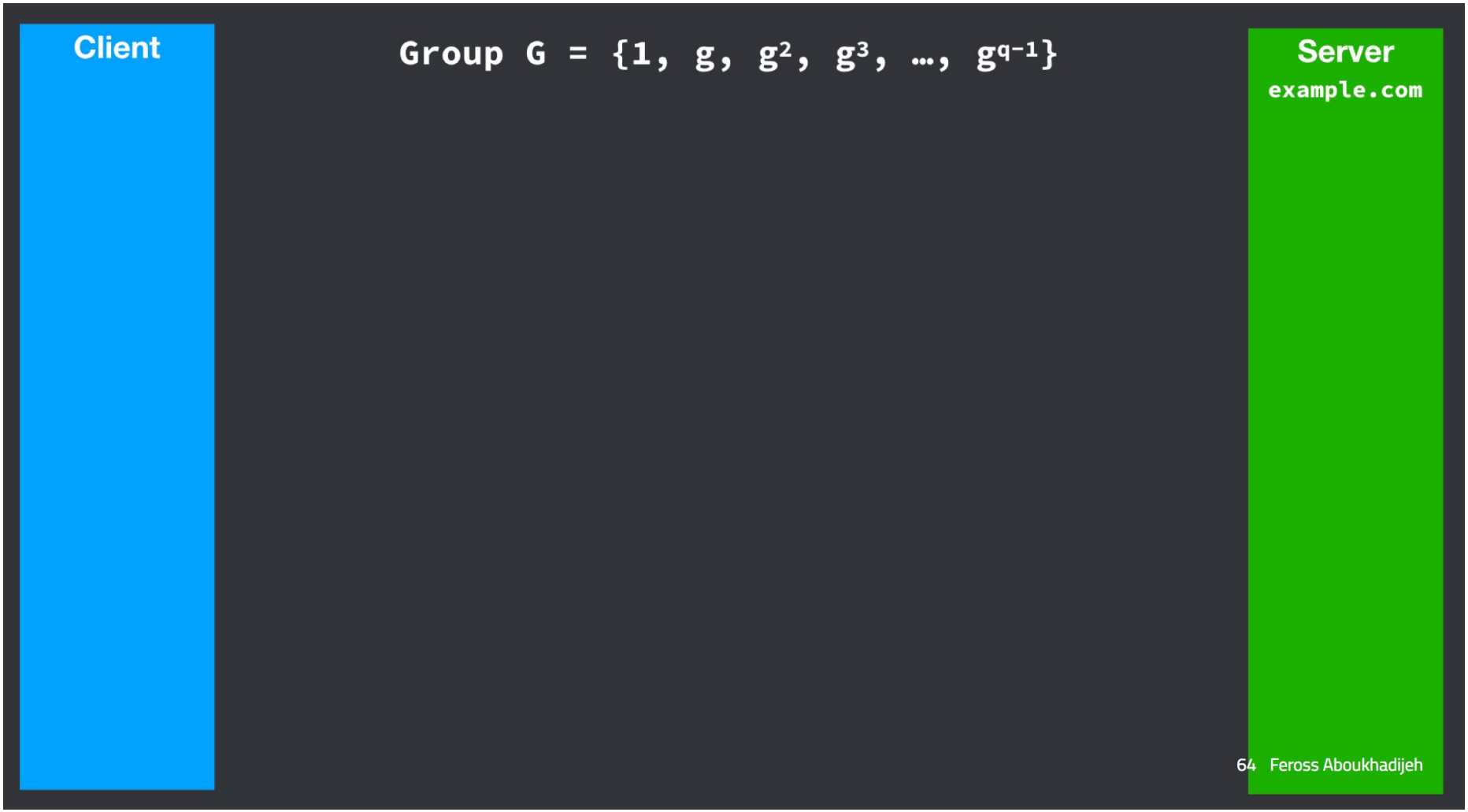
- Triple of algorithms (G, S, V)
  - G() → (pk, sk) - generator returns public key and secret key
  - S(sk, x) → t - signing returns a tag t for input x
  - V(pk, x, t) → accept|reject - checks validity of tag t for given input x
- Algorithm properties
  - Correctness property: V(pk, x, S(sk, x)) = accept should always be true
  - Security property: V(pk, x, t) = accept should almost never be true when x and t are chosen by the attacker

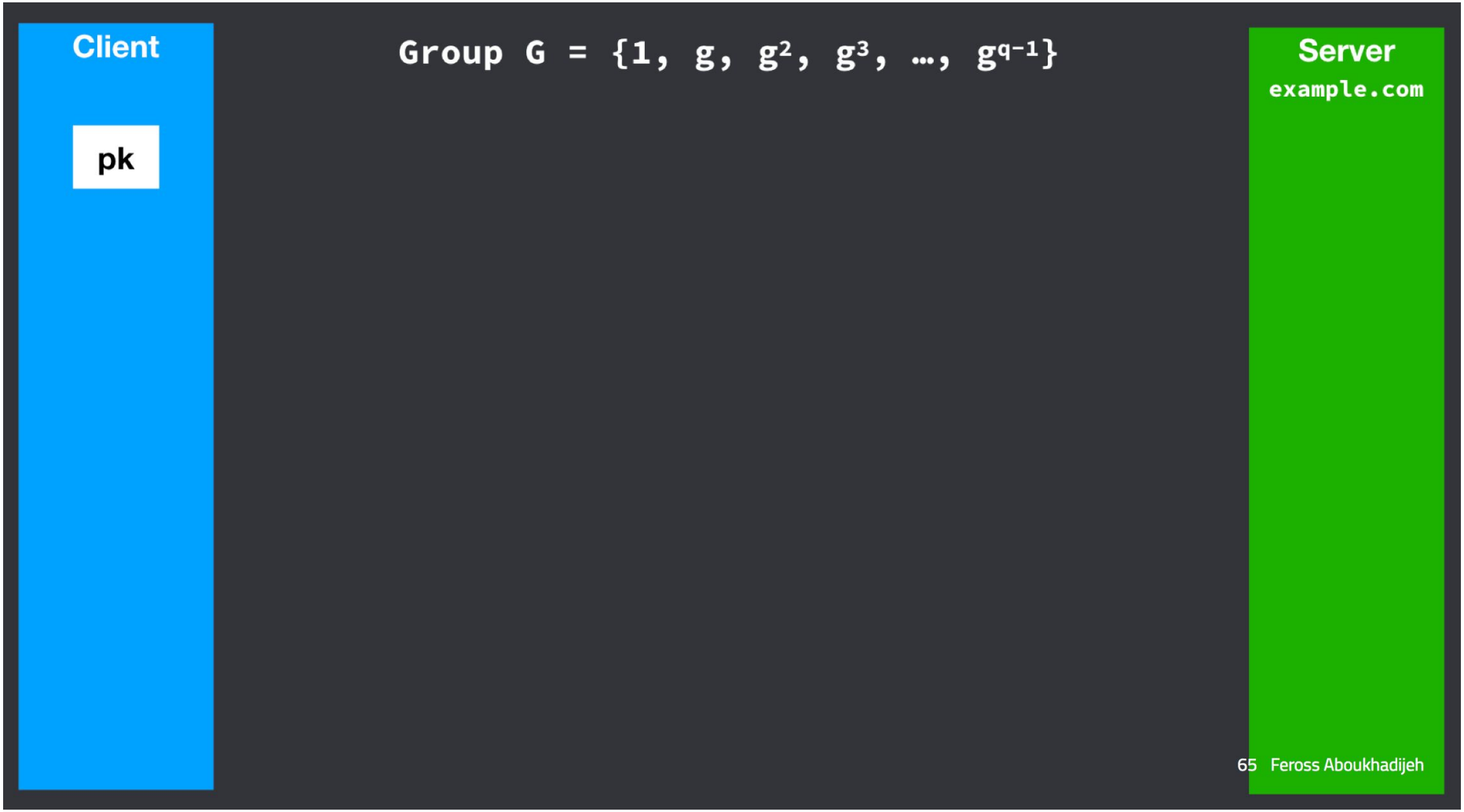
# Authenticated Diffie-Hellman key exchange

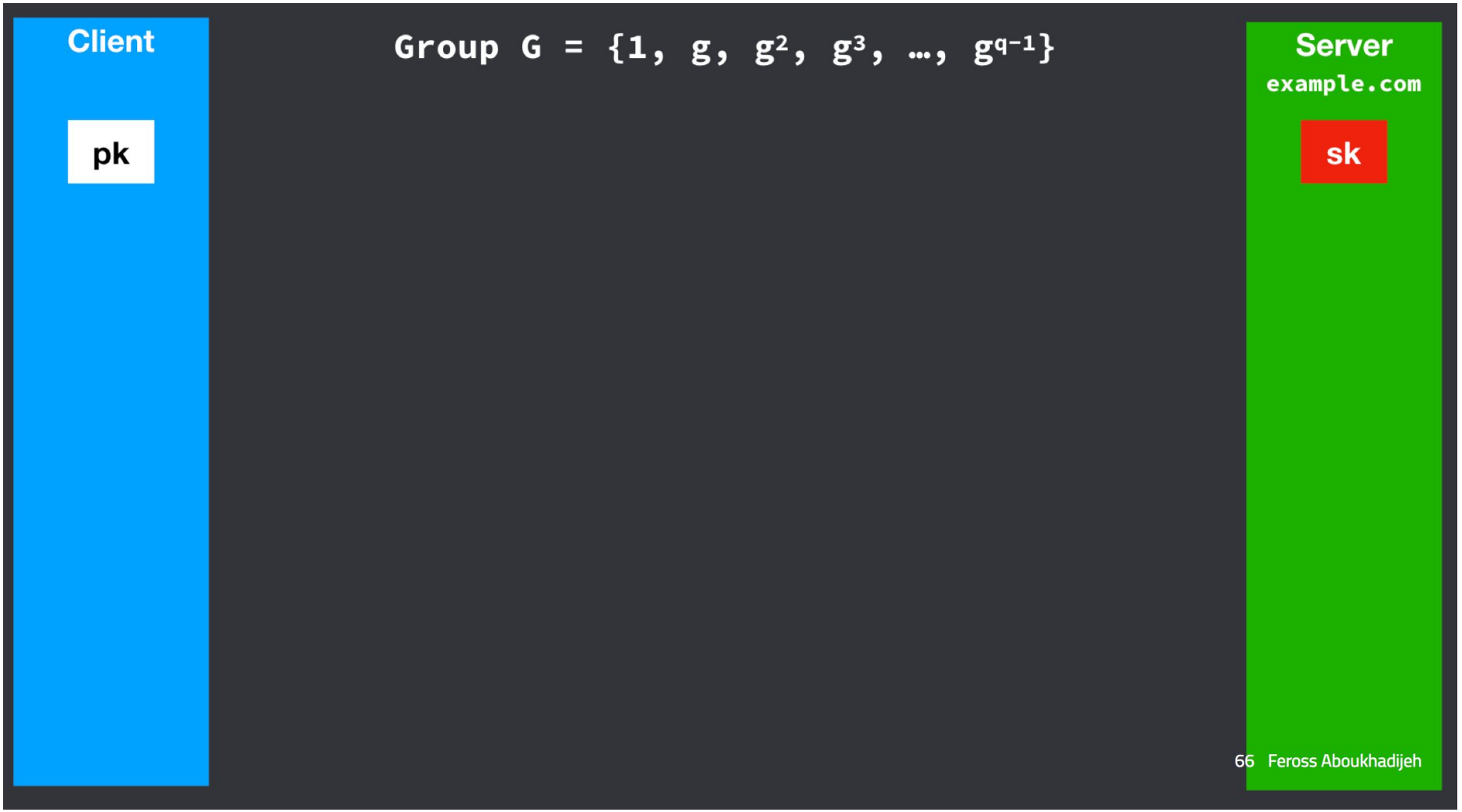
Client

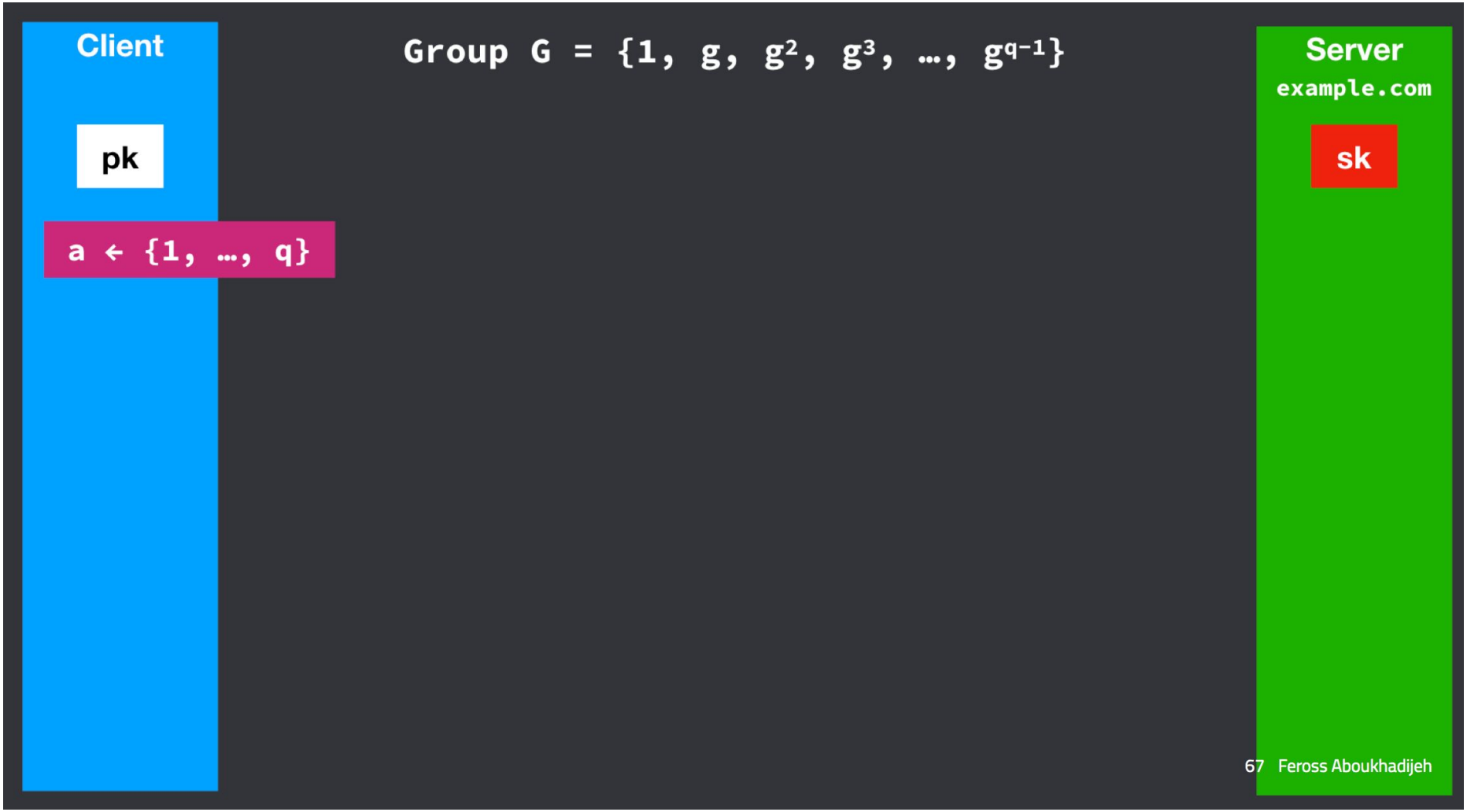
Server
example.com

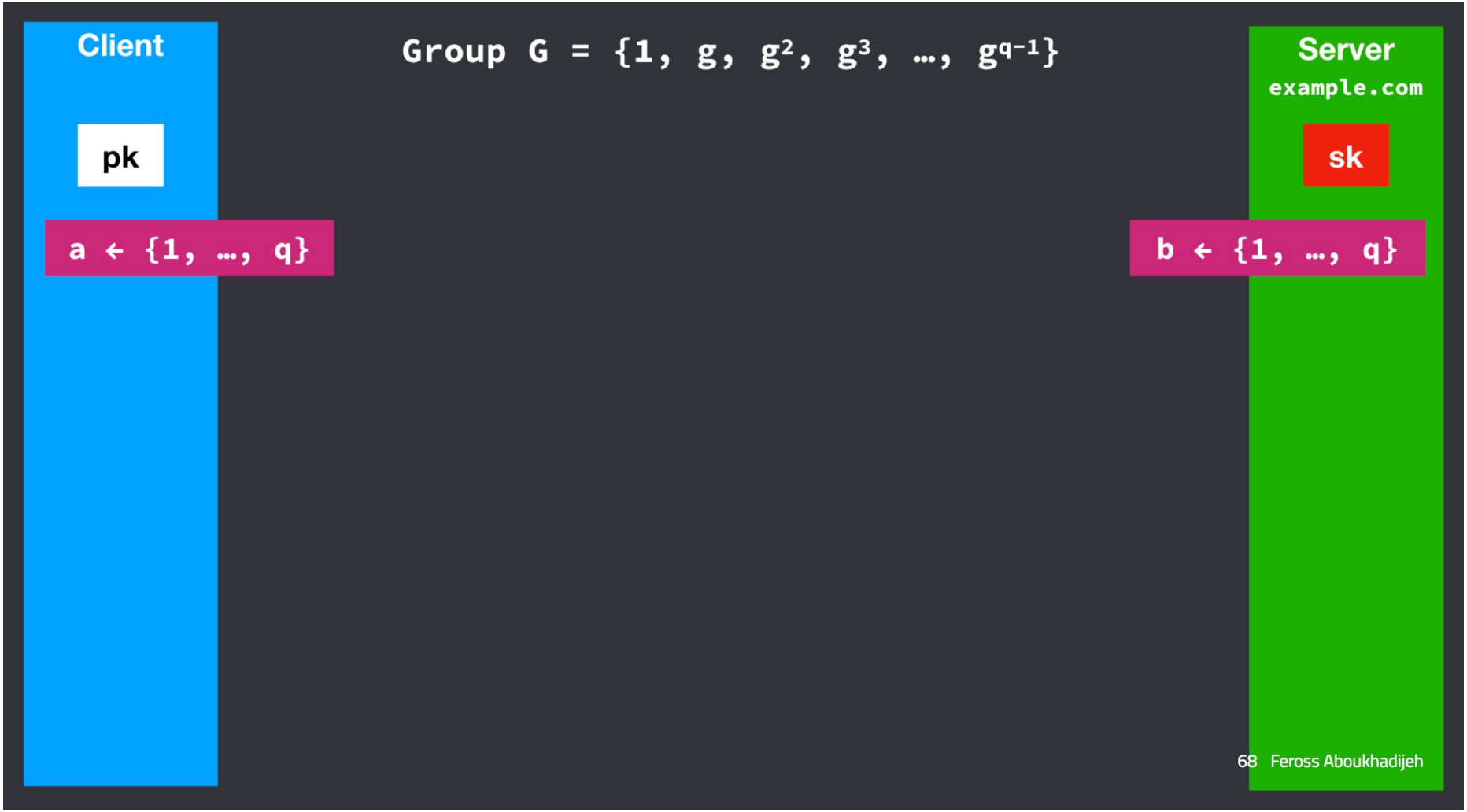

Client
Server
example.com
63 Feross Aboukhadijeh

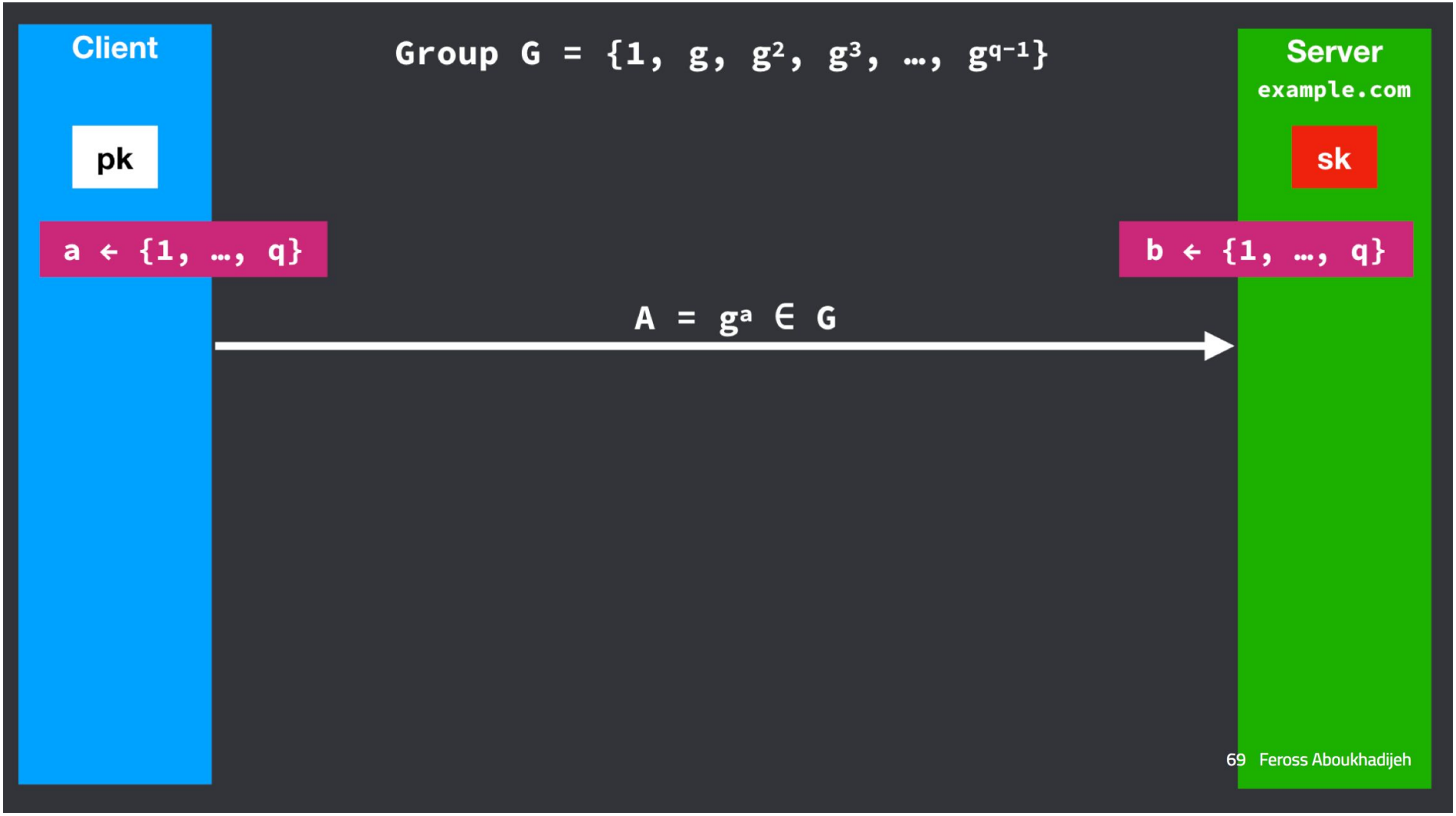

Client
Group G = {1, g, g^2, g^3, …, g^(q-1)}
Server
example.com
64 Feross Aboukhadijeh

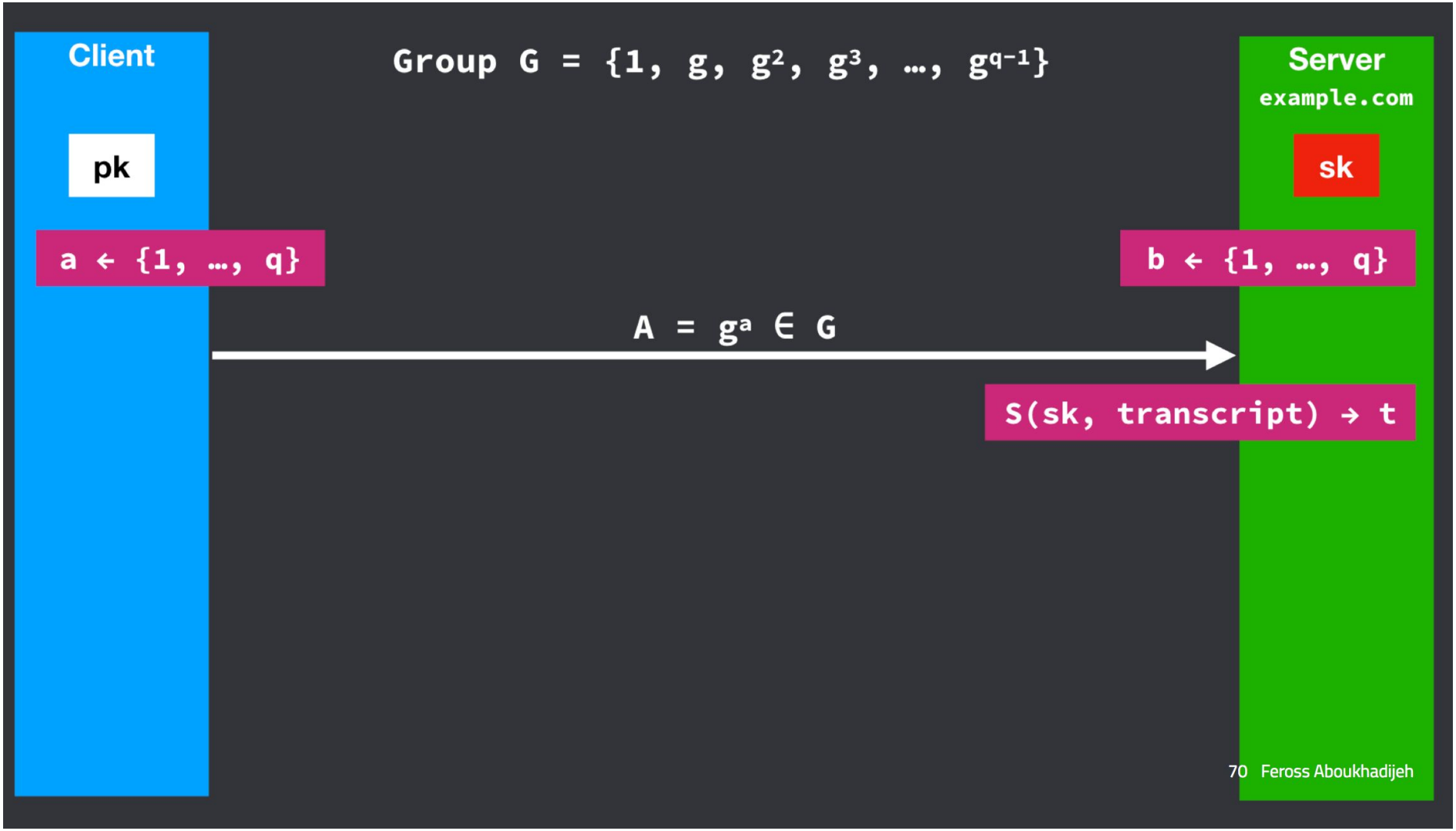

Client
pk
Group G = {1, g, g^2, g^3, …, g^q-1}
Server
example.com
65 Feross Aboukhadijeh

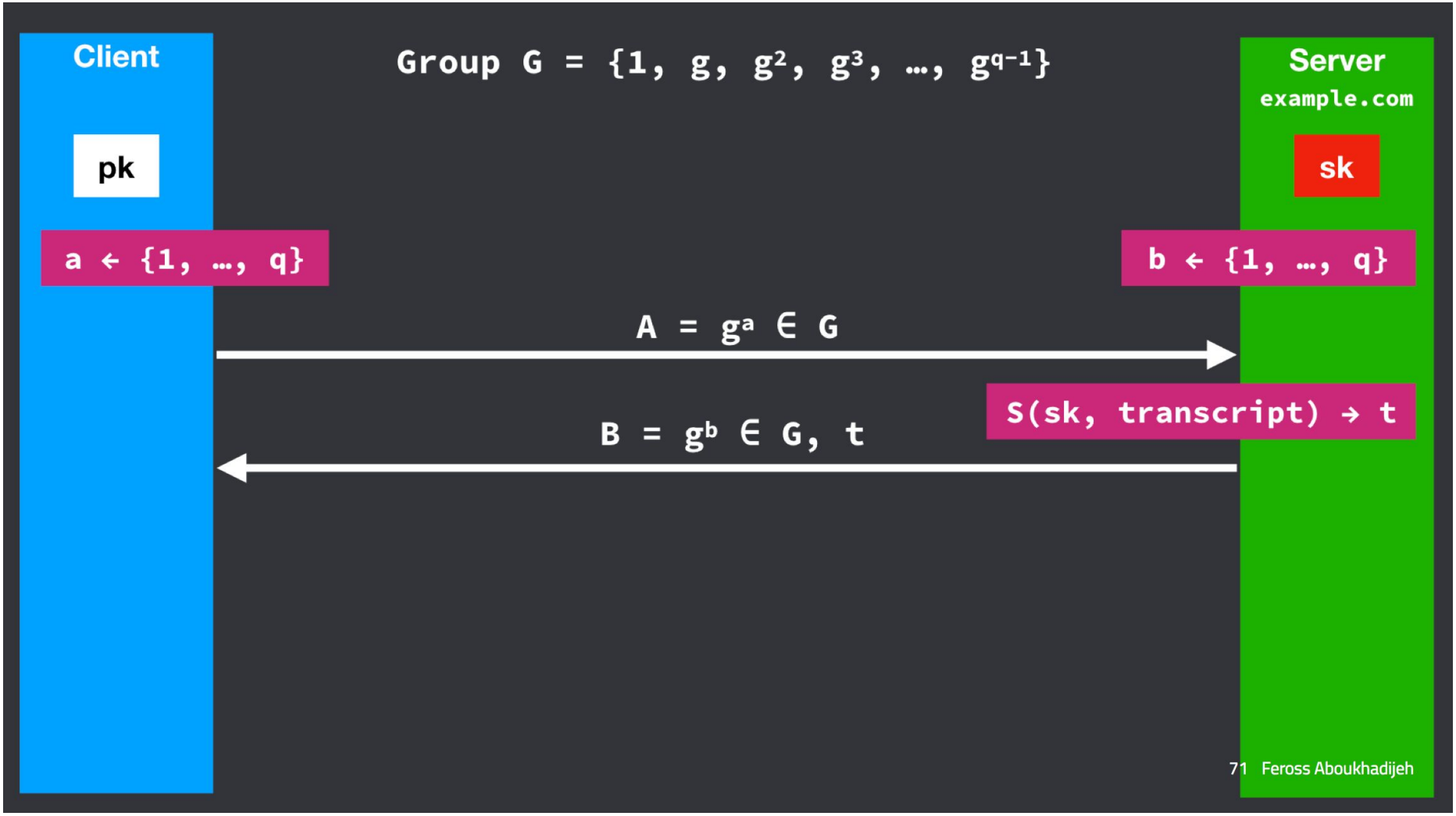

Client
pk
Group G = {1, g, g^2, g^3, …, g^(q-1)}
Server
example.com
sk
66 Feross Aboukhadijeh

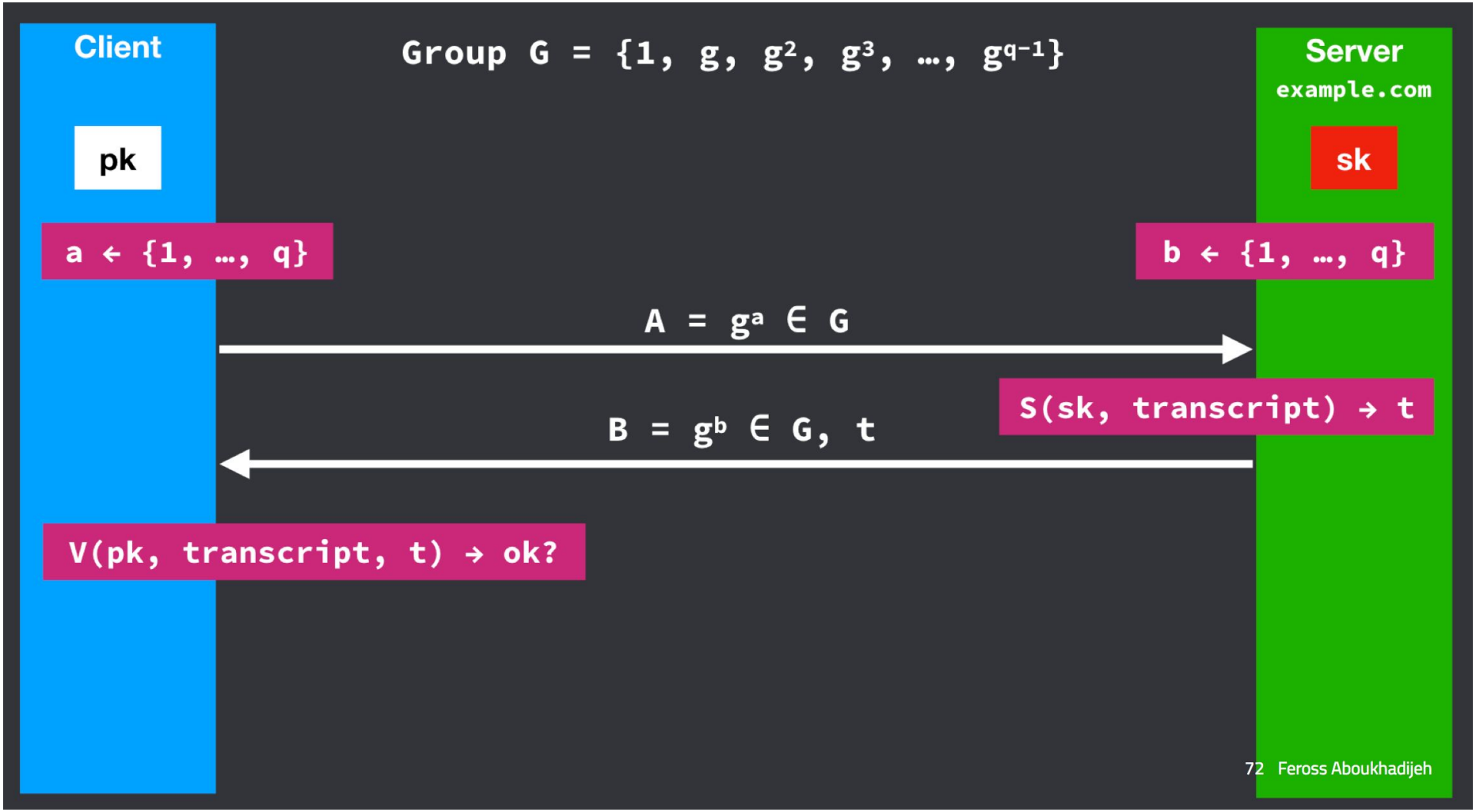

Client
pk
a ← {1, …, q}
Group G = {1, g, g^2, g^3, …, g^(q-1)}
Server
example.com
sk
67 Feross Aboukhadijeh

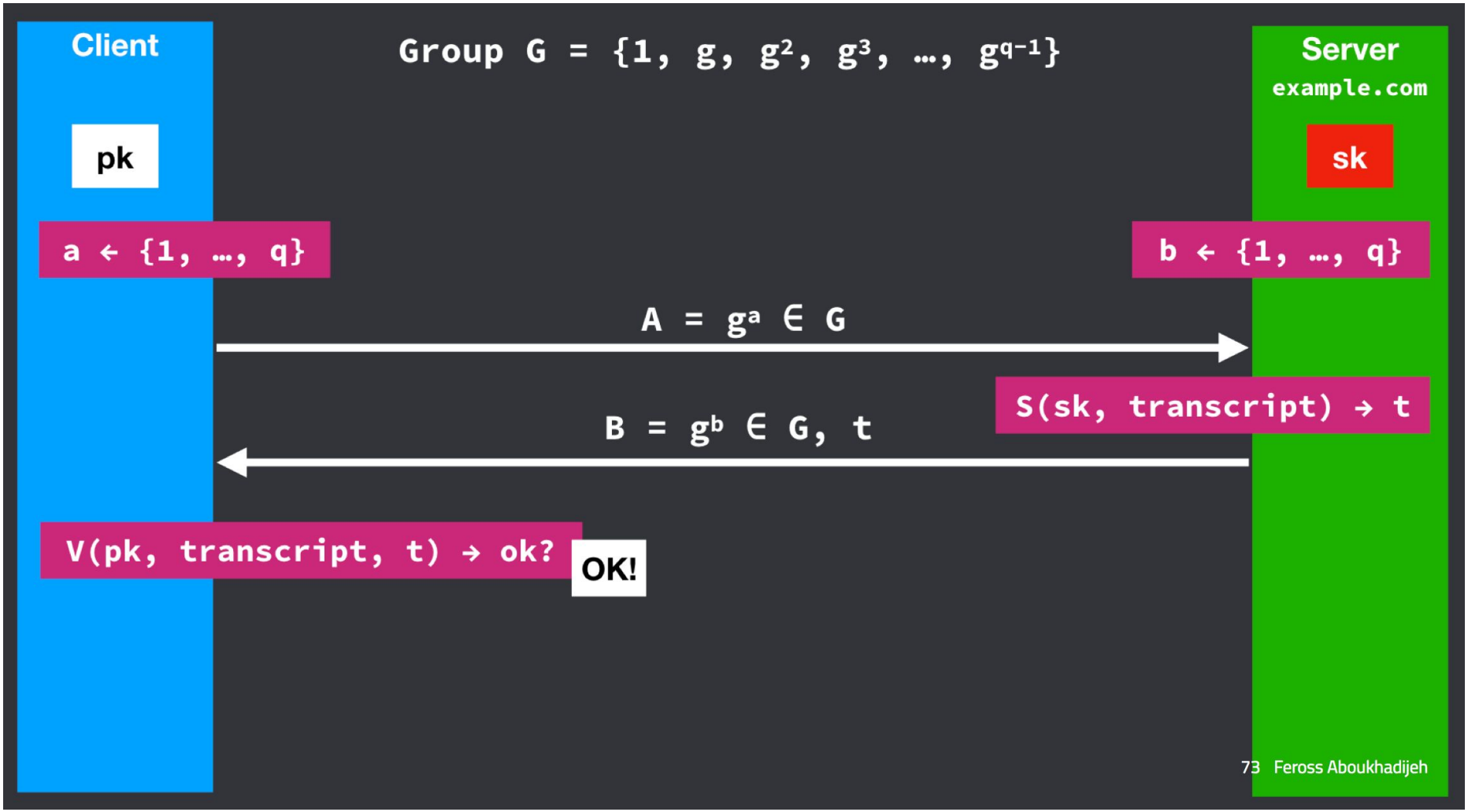

Client
pk
a ← {1, …, q}
Group G = {1, g, g^2, g^3, …, g^(q-1)}
Server
example.com
sk
b ← {1, …, q}
68 Feross Aboukhadijeh

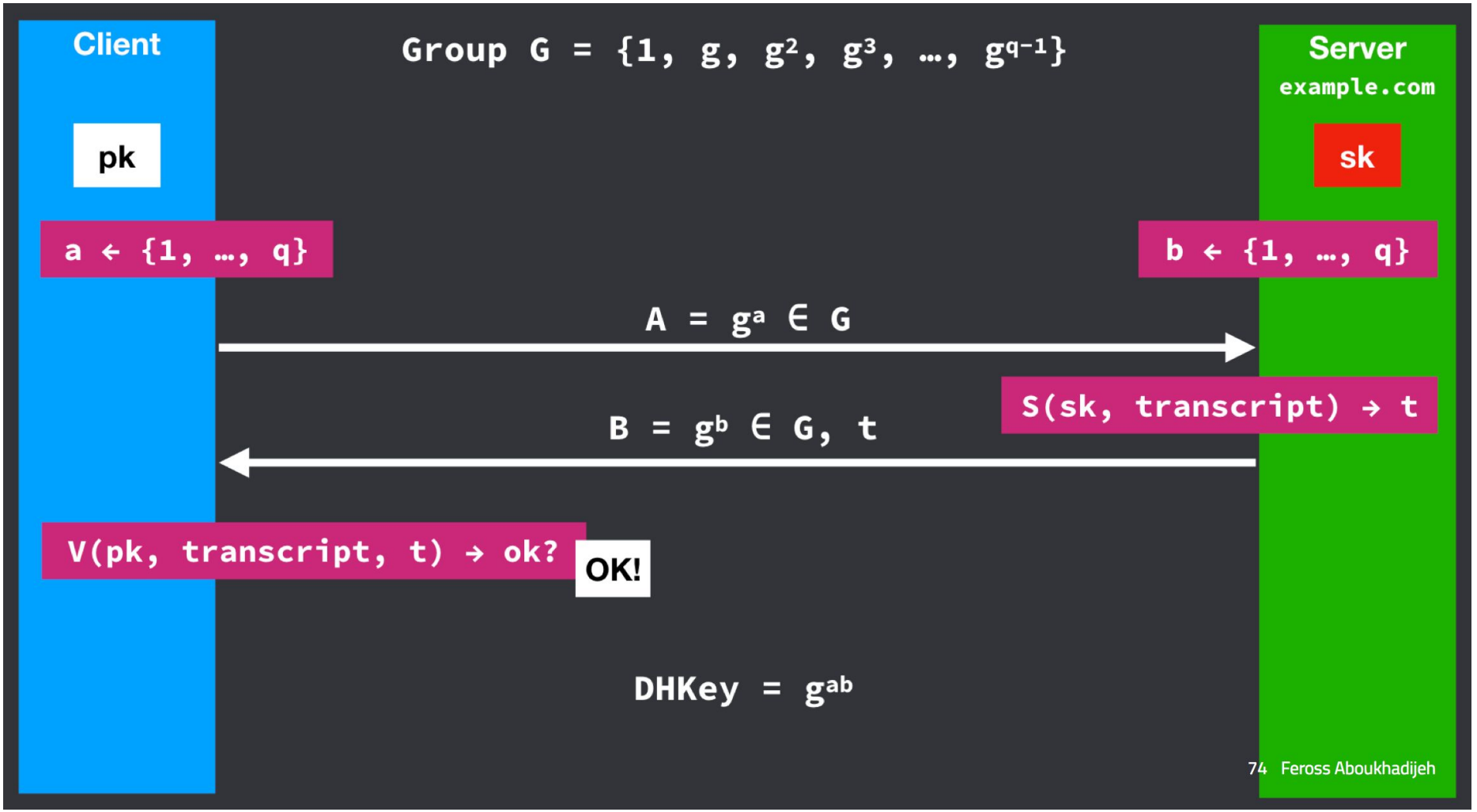

Client
pk
a ← {1, …, q}
Group G = {1, g, g^2, g^3, …, g^(q-1)}
A = g^a ∈ G
Server
example.com
sk
b ← {1, …, q}
69 Feross Aboukhadijeh

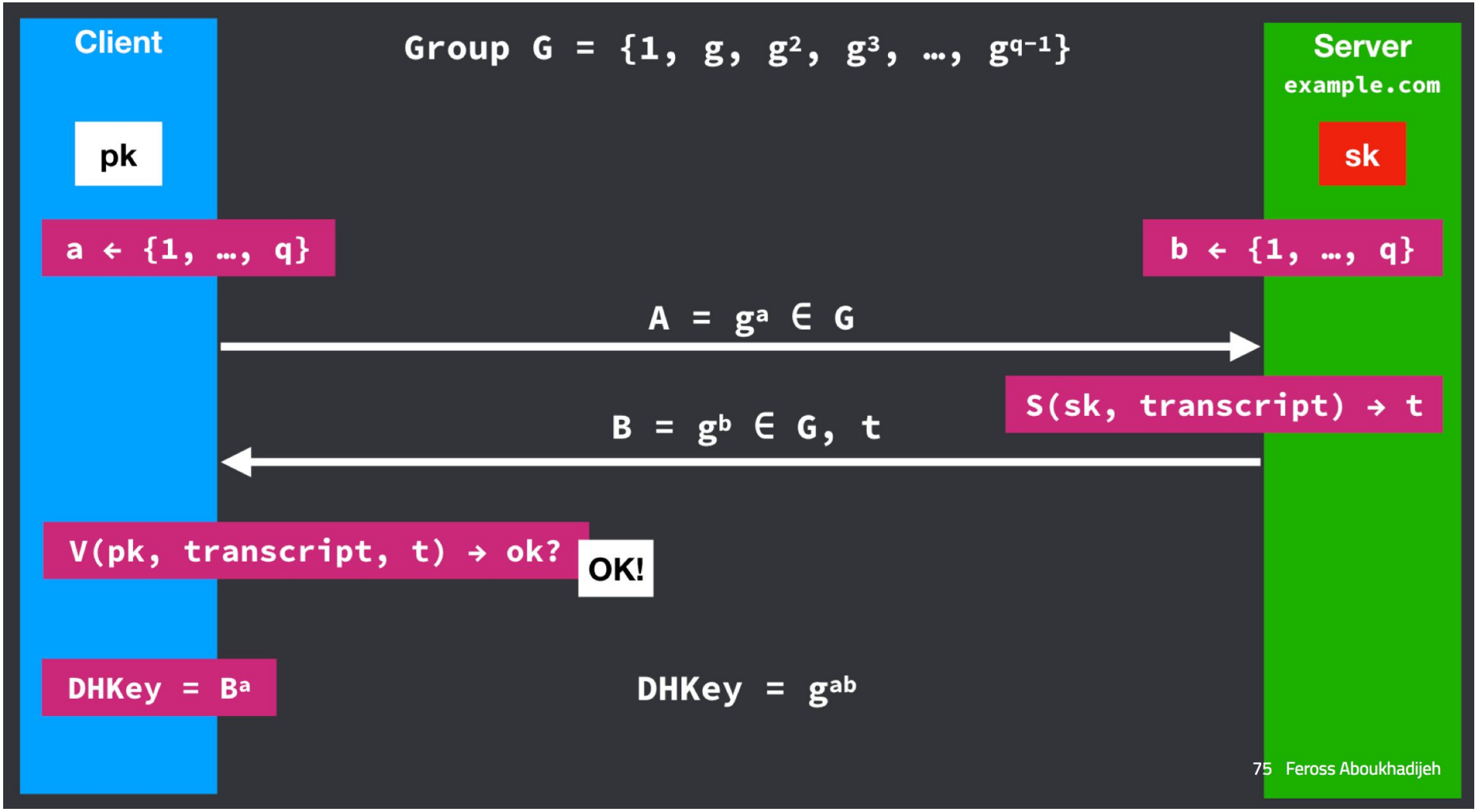

Client
pk
a ← {1, …, q}
Group G = {1, g, g², g³, …, g^(q-1)}
A = g^a ∈ G
Server
example.com
sk
b ← {1, …, q}
S(sk, transcript) → t
70 Feross Aboukhadijeh

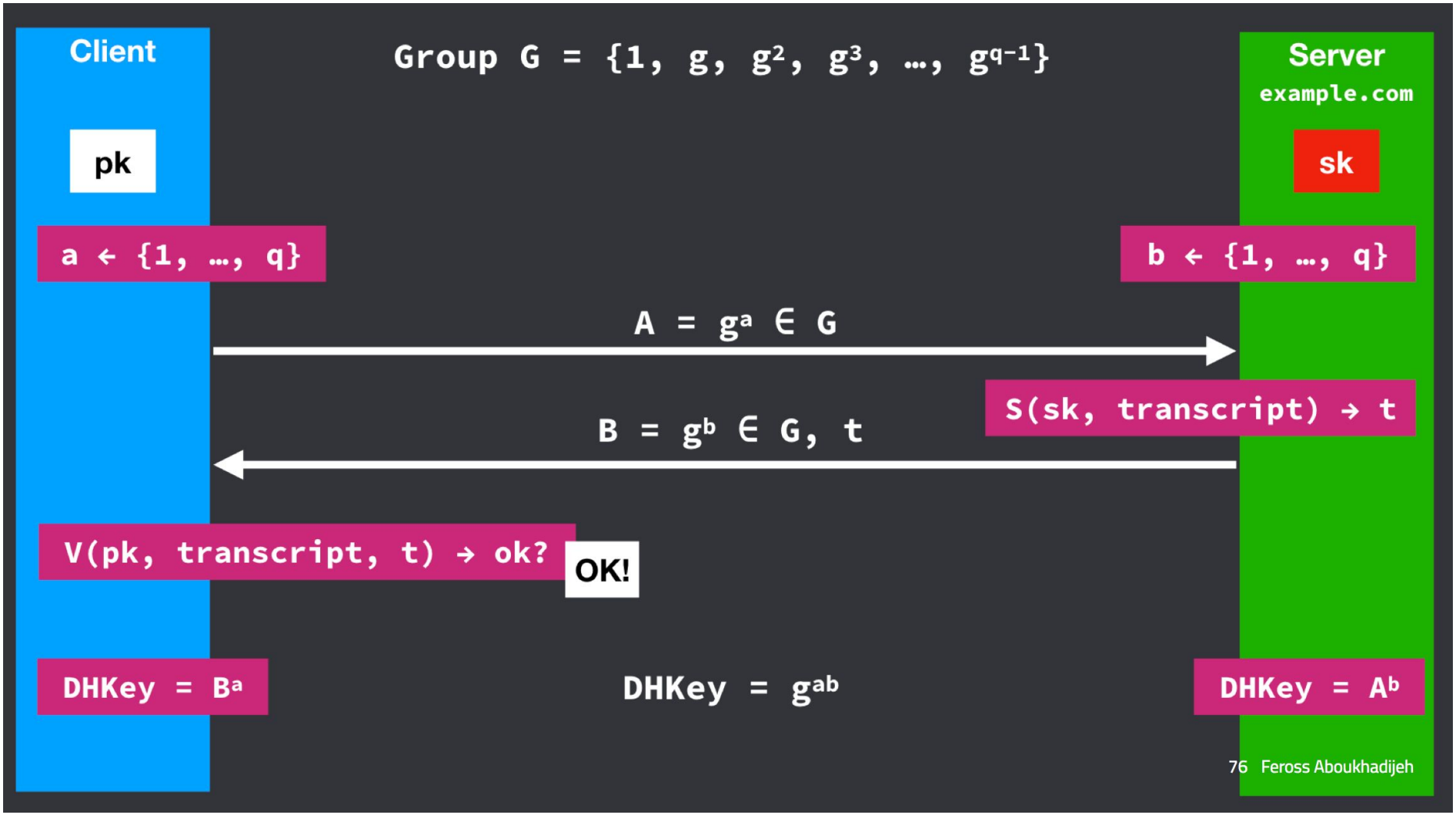

Client
pk
a ← {1, …, q}
Group G = {1, g, g², g³, …, g^(q-1)}
Server
example.com
sk
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
71 Feross Aboukhadijeh

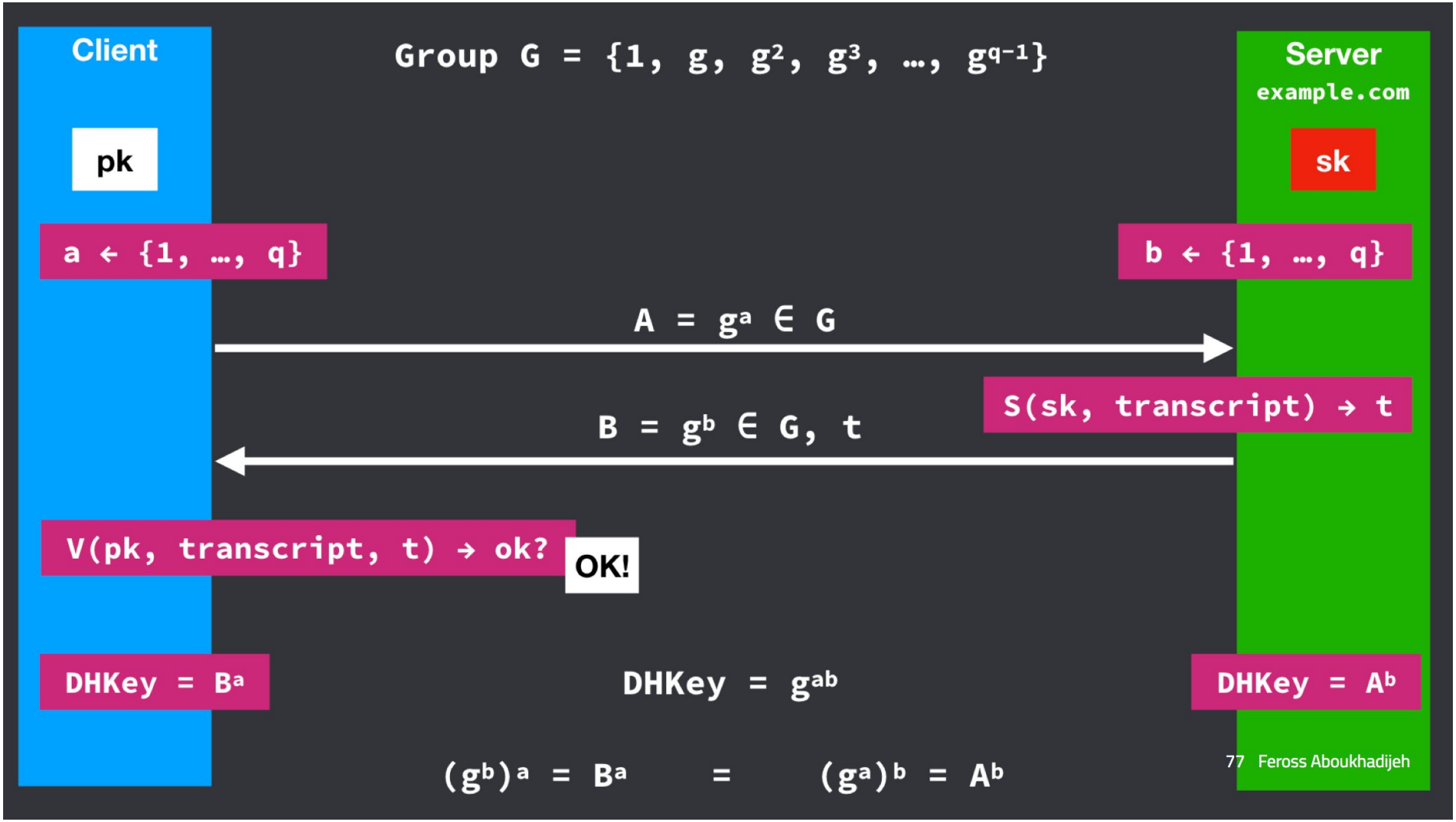

Client
pk
Group G = {1, g, g², g³, …, g^(q-1)}
Server
example.com
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
72 Feross Aboukhadijeh


Client
pk
Group G = {1, g, g^2, g^3, …, g^(q-1)}
Server
example.com
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
OK!
73 Feross Aboukhadijeh


Client
Group G = {1, g, g², g³, …, g^(q-1)}
Server
example.com
pk
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
OK!
DHKey = g^ab
74 Feross Aboukhadijeh

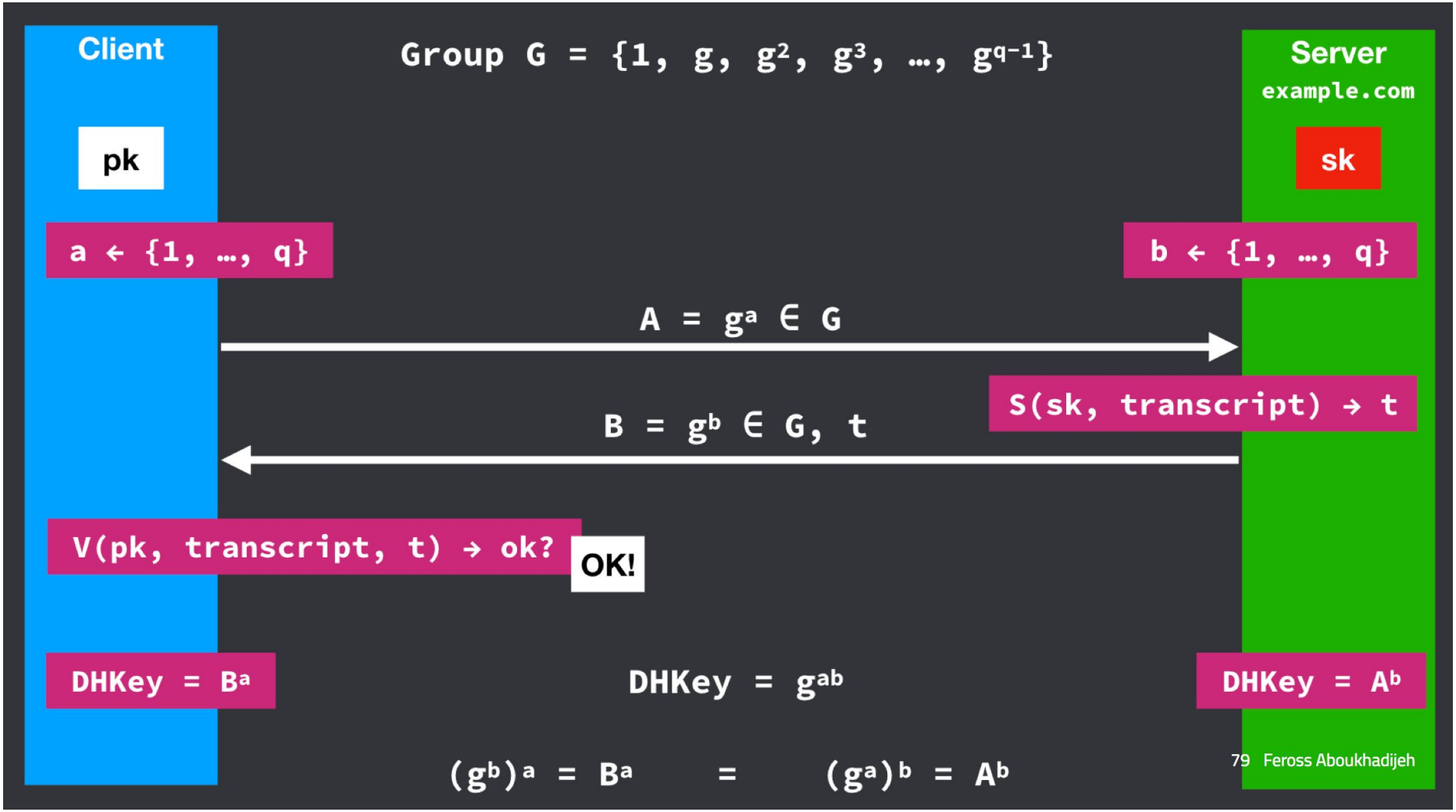

Client
Group G = {1, g, g², g³, …, g^(q-1)}
Server
example.com
pk
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
OK!
DHKey = B^a
DHKey = g^ab
75 Feross Aboukhadijeh


Client
Group G = {1, g, g^2, g^3, …, g^(q-1)}
Server
example.com
pk
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
OK!
DHKey = B^a
DHKey = g^ab
DHKey = A^b
76 Feross Aboukhadijeh


Client
Group G = {1, g, g², g³, …, g^(q-1)}
Server
example.com
pk
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
OK!
DHKey = B^a
DHKey = g^ab
DHKey = A^b
(g^b)^a = B^a = (g^a)^b = A^b
77 Feross Aboukhadijeh

# How does the client get the server's public key?

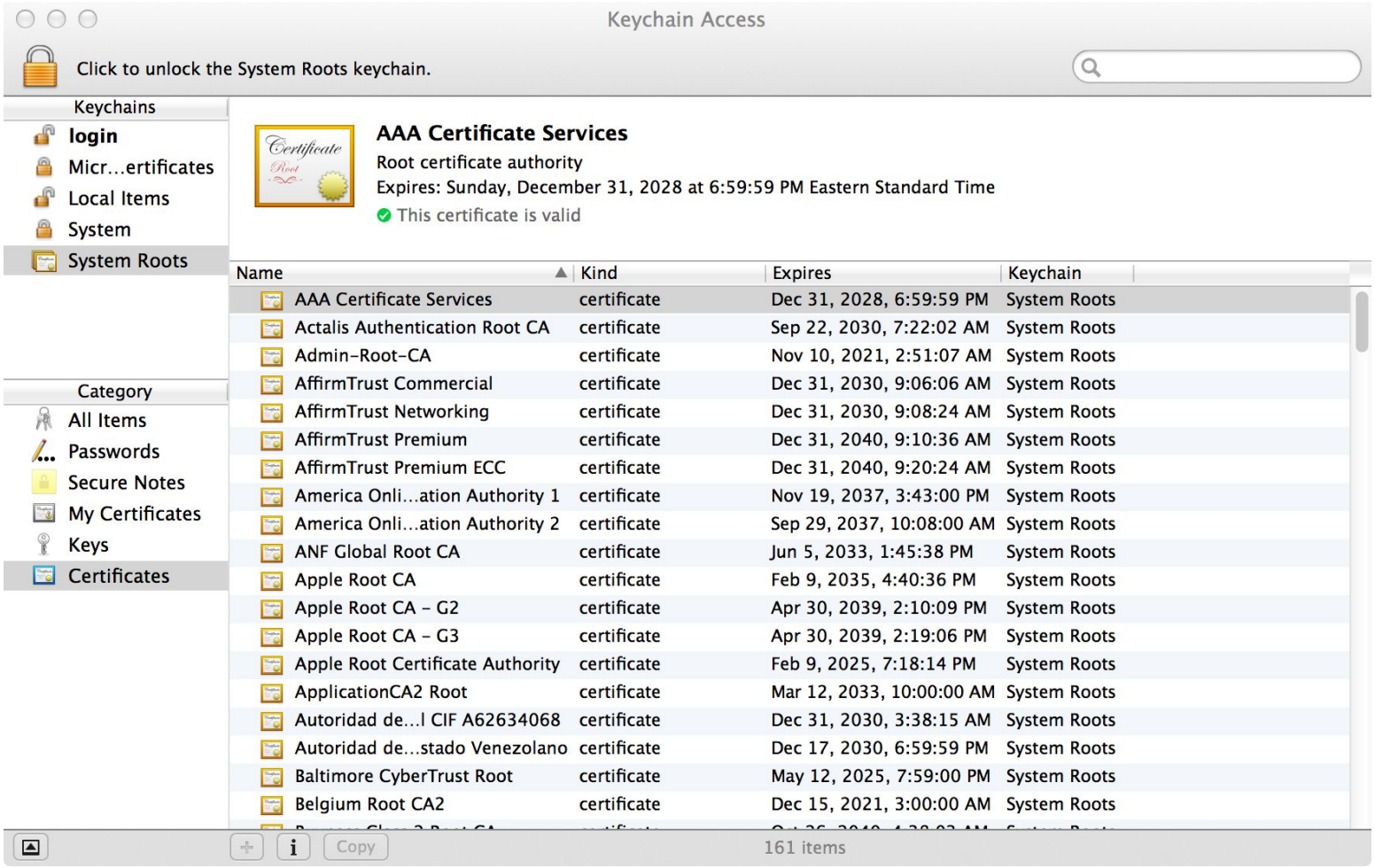
- Idea: Build in every website's public key into the browser
  - Would be a huge list, constantly changing, cannot connect to server if list is out-of-date
- Idea: Server can send the public key to the client during the key exchange
  - Back to the same problem as anonymous key exchange!
  - What's to stop an active network attacker from sending their own public key in the exchange?

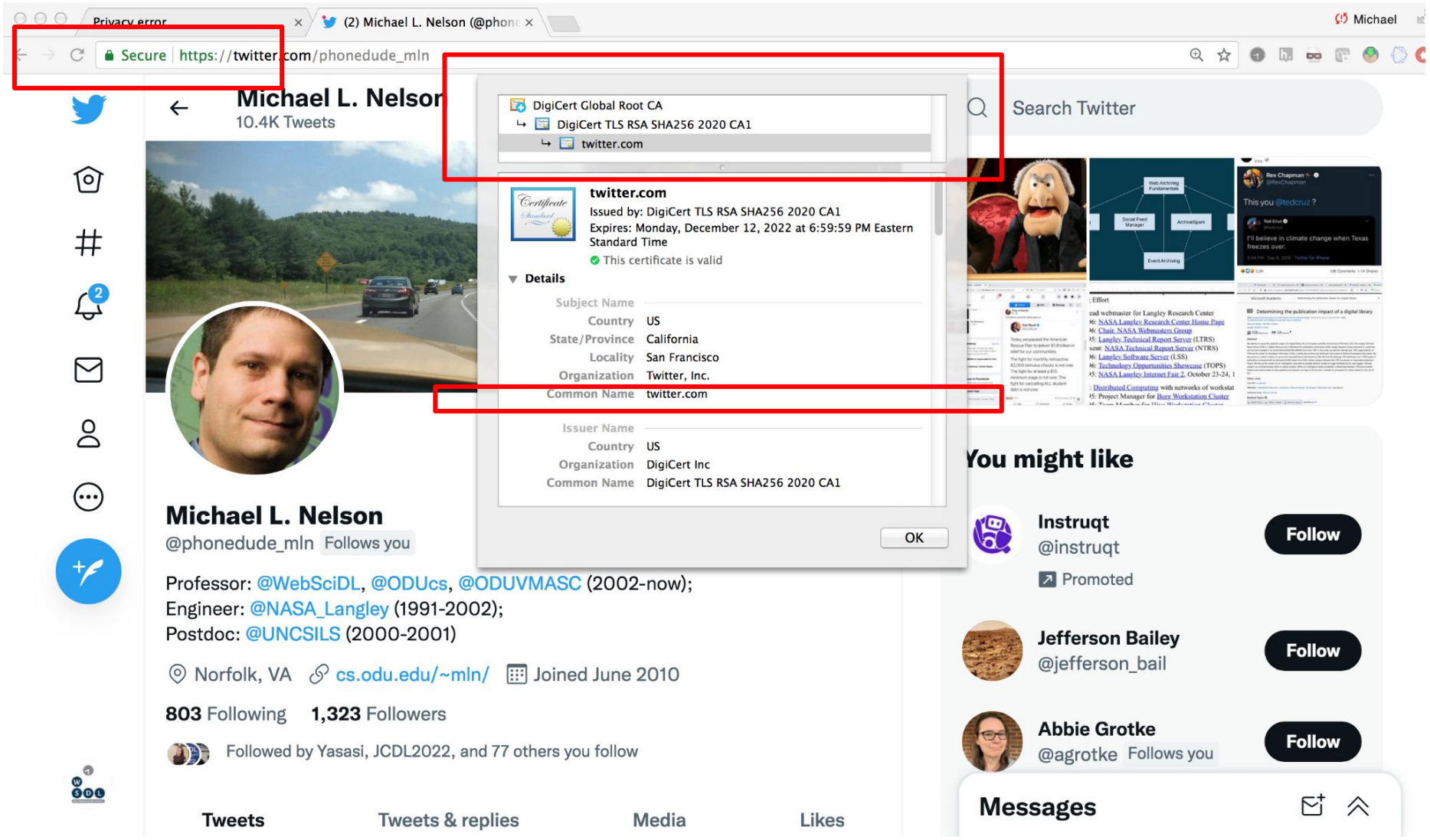

Client
Group G = {1, g, g², g³, …, g^(q-1)}
Server
example.com
pk
sk
a ← {1, …, q}
b ← {1, …, q}
A = g^a ∈ G
S(sk, transcript) → t
B = g^b ∈ G, t
V(pk, transcript, t) → ok?
OK!
DHKey = B^a
DHKey = g^ab
DHKey = A^b
(g^b)^a = B^a = (g^a)^b = A^b
79 Feross Aboukhadijeh

# Certificate authorities (CAs)

- A certificate authority (CA) is an entity that issues digital certificates
- A certificate certifies that a named subject is the owner of a specific public key
  - "I, CERTIFICATE_AUTHORITY, certify that SUBJECT_NAME is the owner of public key PUBLIC_KEY"

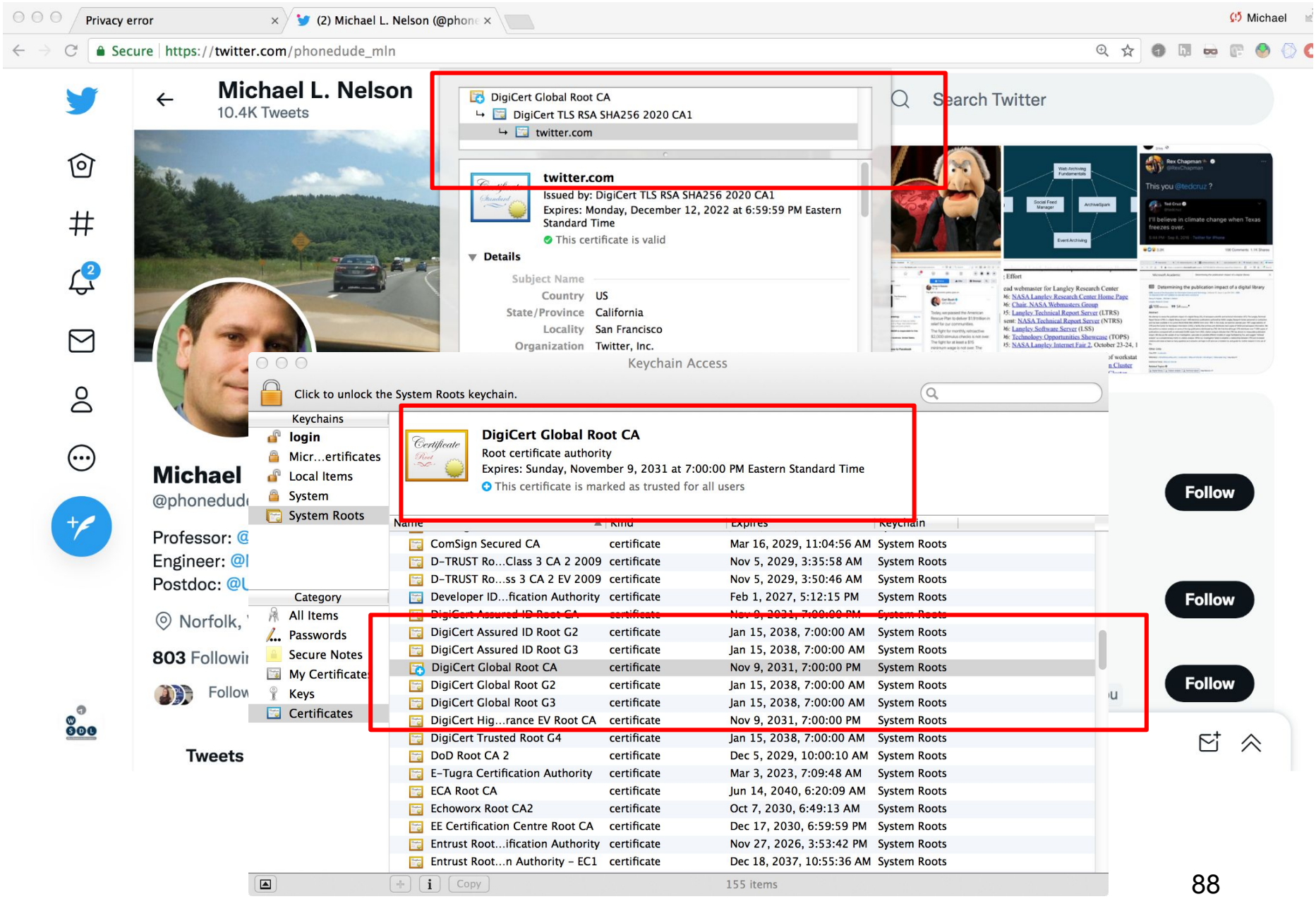
# Who does your browser trust?

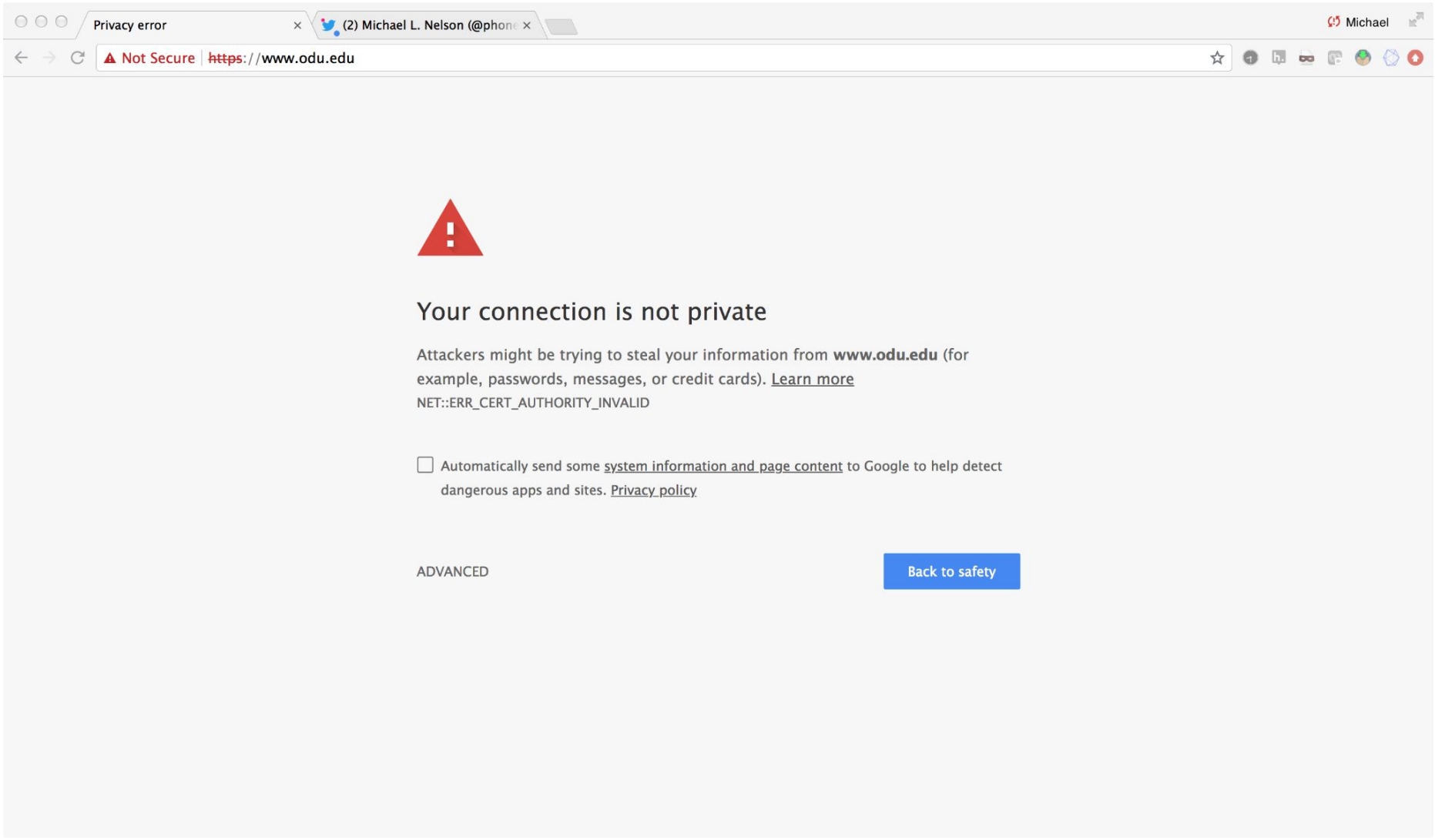


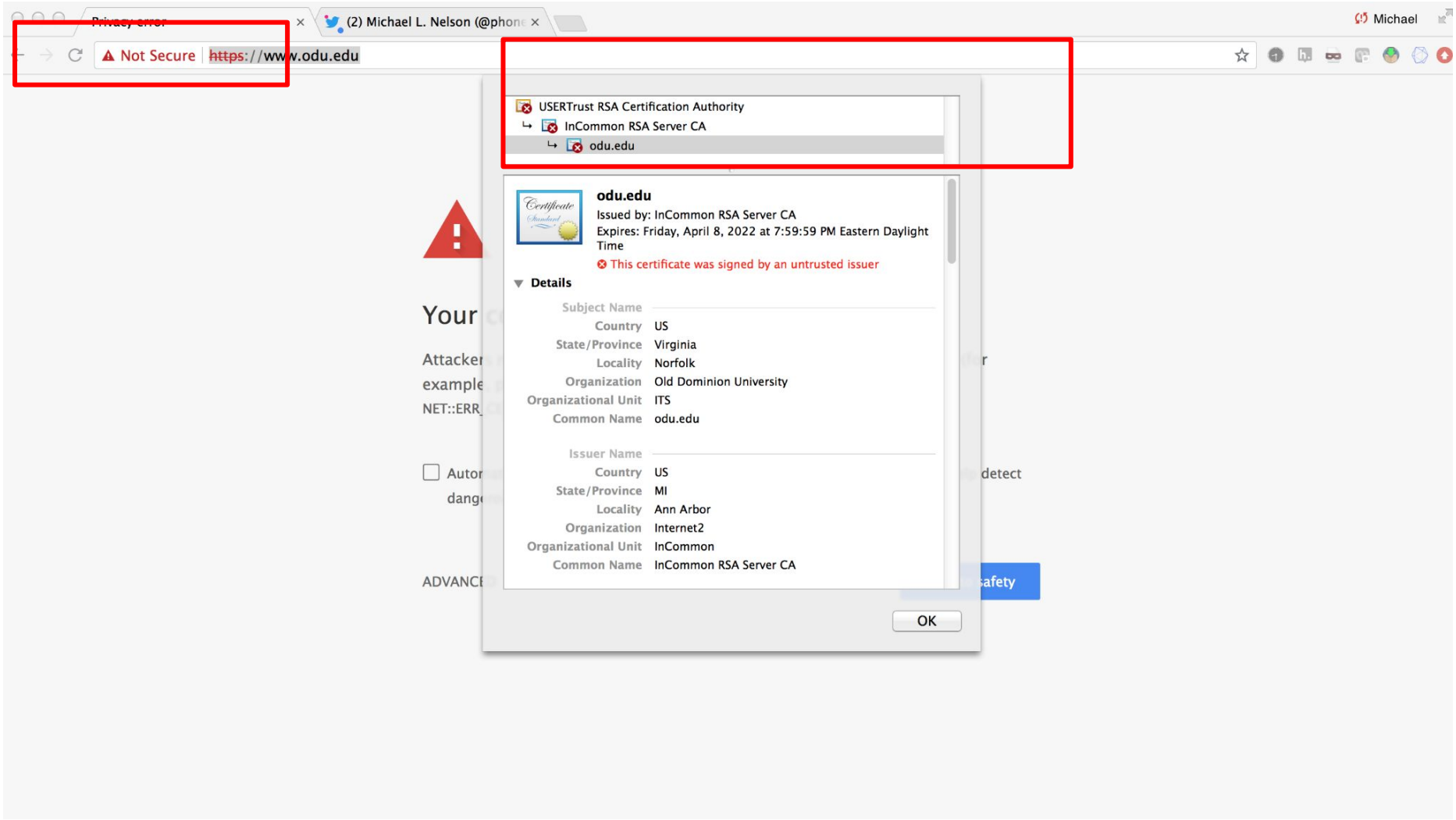

Privacy error
(2) Michael L. Nelson (@phone
Michael
Secure | https://twitter.com/phonedude_mln
Michael L. Nelson
10.4K Tweets
DigiCert Global Root CA
DigiCert TLS RSA SHA256 2020 CA1
twitter.com
twitter.com
Issued by: DigiCert TLS RSA SHA256 2020 CA1
Expires: Monday, December 12, 2022 at 6:59:59 PM Eastern Standard Time
This certificate is valid
Details
Subject Name
Country US
State/Province California
Locality San Francisco
Organization Twitter, Inc.
Common Name twitter.com
Issuer Name
Country US
Organization DigiCert Inc
Common Name DigiCert TLS RSA SHA256 2020 CA1
OK
Search Twitter
Michael L. Nelson
@phonedude_mln Follows you
Professor: @WebSciDL, @ODUcs, @ODUVMASC (2002-now);
Engineer: @NASA_Langley (1991-2002);
Postdoc: @UNCSILS (2000-2001)
Norfolk, VA cs.odu.edu/~mln/ Joined June 2010
803 Following 1,323 Followers
Followed by Yasasi, JCDL2022, and 77 others you follow
Tweets
Tweets & replies
Media
Likes
You might like
Instruqt
@instruqt
Promoted
Follow
Jefferson Bailey
@jefferson_bail
Follow
Abbie Grotke
@agrotke Follows you
Follow
Messages

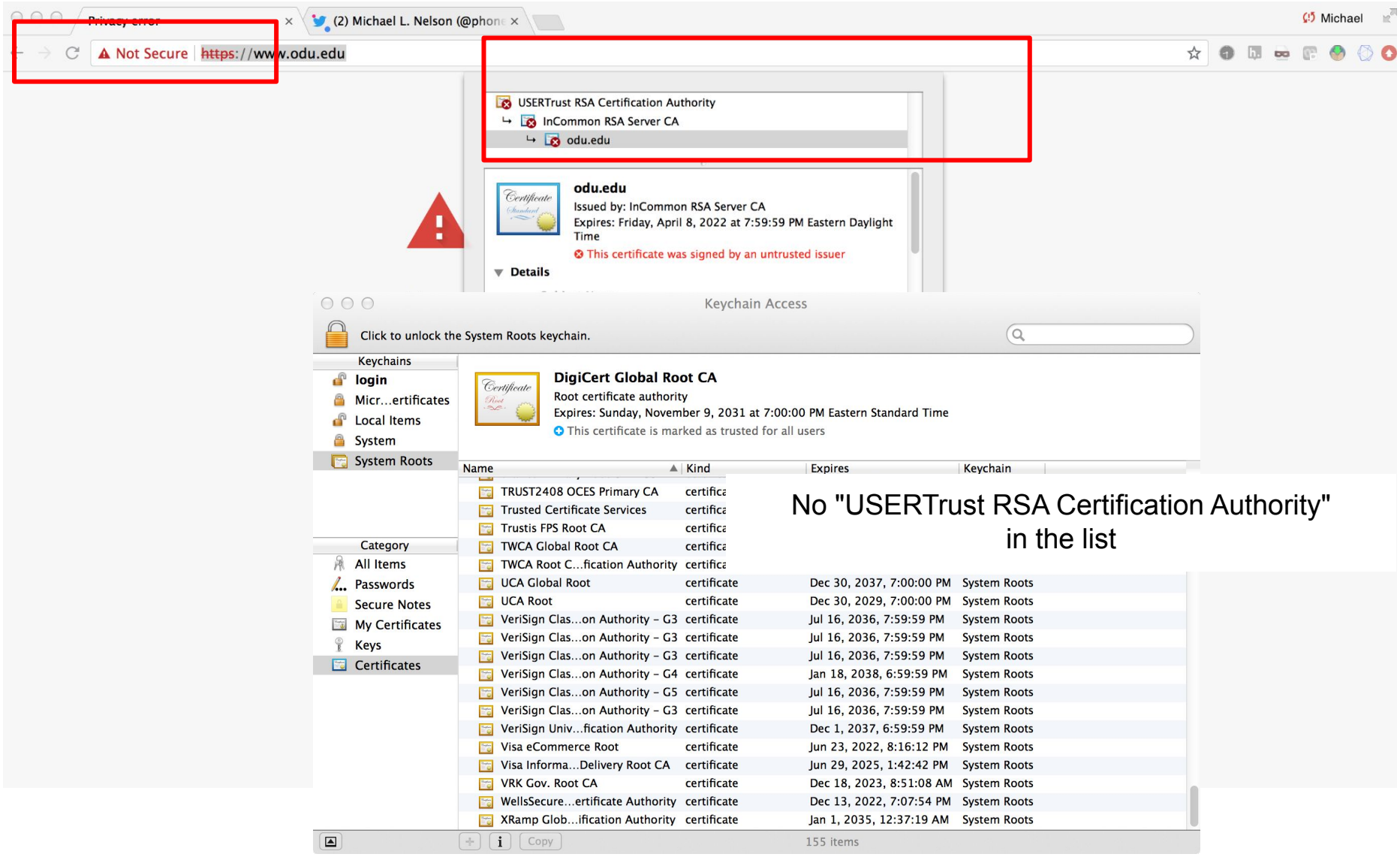
# Common name rules

- Subject's CommonName can be:
  - an explicit name, e.g. cs.odu.edu
  - a wildcard cert, e.g. *.odu.edu or cs*.odu.edu
- Matching rules
  - The * must occur in the leftmost subdomain component
  - The * does not match . characters
  - Example: *.odu.edu matches cs.odu.edu but not memgator.cs.odu.edu

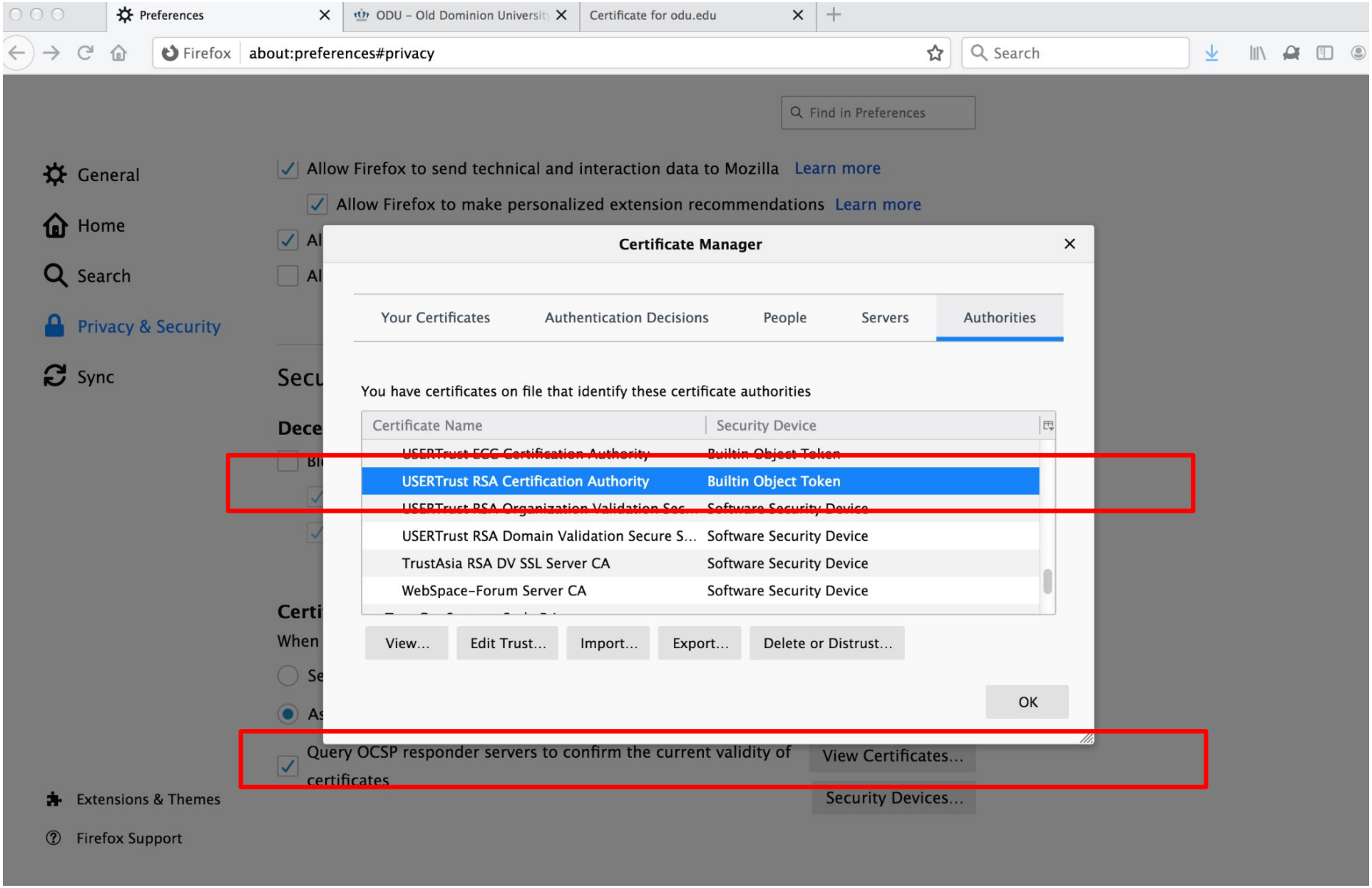

Privacy error
(2) Michael L. Nelson (@phone
Michael
Secure | https://twitter.com/phonedude_mln
Michael L. Nelson
10.4K Tweets
Search Twitter
DigiCert Global Root CA
DigiCert TLS RSA SHA256 2020 CA1
twitter.com
twitter.com
Issued by: DigiCert TLS RSA SHA256 2020 CA1
Expires: Monday, December 12, 2022 at 6:59:59 PM Eastern Standard Time
This certificate is valid
Details
Subject Name
Country US
State/Province California
Locality San Francisco
Organization Twitter, Inc.
Keychain Access
Click to unlock the System Roots keychain.
Keychains
login
Micr...ertificates
Local Items
System
System Roots
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
DigiCert Global Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 7:00:00 PM Eastern Standard Time
This certificate is marked as trusted for all users
Name
Kind
Expires
Keychain
ComSign Secured CA certificate Mar 16, 2029, 11:04:56 AM System Roots
D-TRUST Ro...Class 3 CA 2 2009 certificate Nov 5, 2029, 3:35:58 AM System Roots
D-TRUST Ro...ss 3 CA 2 EV 2009 certificate Nov 5, 2029, 3:50:46 AM System Roots
Developer ID...fication Authority certificate Feb 1, 2027, 5:12:15 PM System Roots
DigiCert Assured ID Root CA certificate Nov 9, 2031, 7:00:00 PM System Roots
DigiCert Assured ID Root G2 certificate Jan 15, 2038, 7:00:00 AM System Roots
DigiCert Assured ID Root G3 certificate Jan 15, 2038, 7:00:00 AM System Roots
DigiCert Global Root CA certificate Nov 9, 2031, 7:00:00 PM System Roots
DigiCert Global Root G2 certificate Jan 15, 2038, 7:00:00 AM System Roots
DigiCert Global Root G3 certificate Jan 15, 2038, 7:00:00 AM System Roots
DigiCert Hig...rance EV Root CA certificate Nov 9, 2031, 7:00:00 PM System Roots
DigiCert Trusted Root G4 certificate Jan 15, 2038, 7:00:00 AM System Roots
DoD Root CA 2 certificate Dec 5, 2029, 10:00:10 AM System Roots
E-Tugra Certification Authority certificate Mar 3, 2023, 7:09:48 AM System Roots
ECA Root CA certificate Jun 14, 2040, 6:20:09 AM System Roots
Echoworx Root CA2 certificate Oct 7, 2030, 6:49:13 AM System Roots
EE Certification Centre Root CA certificate Dec 17, 2030, 6:59:59 PM System Roots
Entrust Root...ification Authority certificate Nov 27, 2026, 3:53:42 PM System Roots
Entrust Root...n Authority - EC1 certificate Dec 18, 2037, 10:55:36 AM System Roots
Copy
155 items
Michael
Professor:
Engineer:
Postdoc:
Norfolk,
803 Followi
Follow
Follow
Follow
Tweets

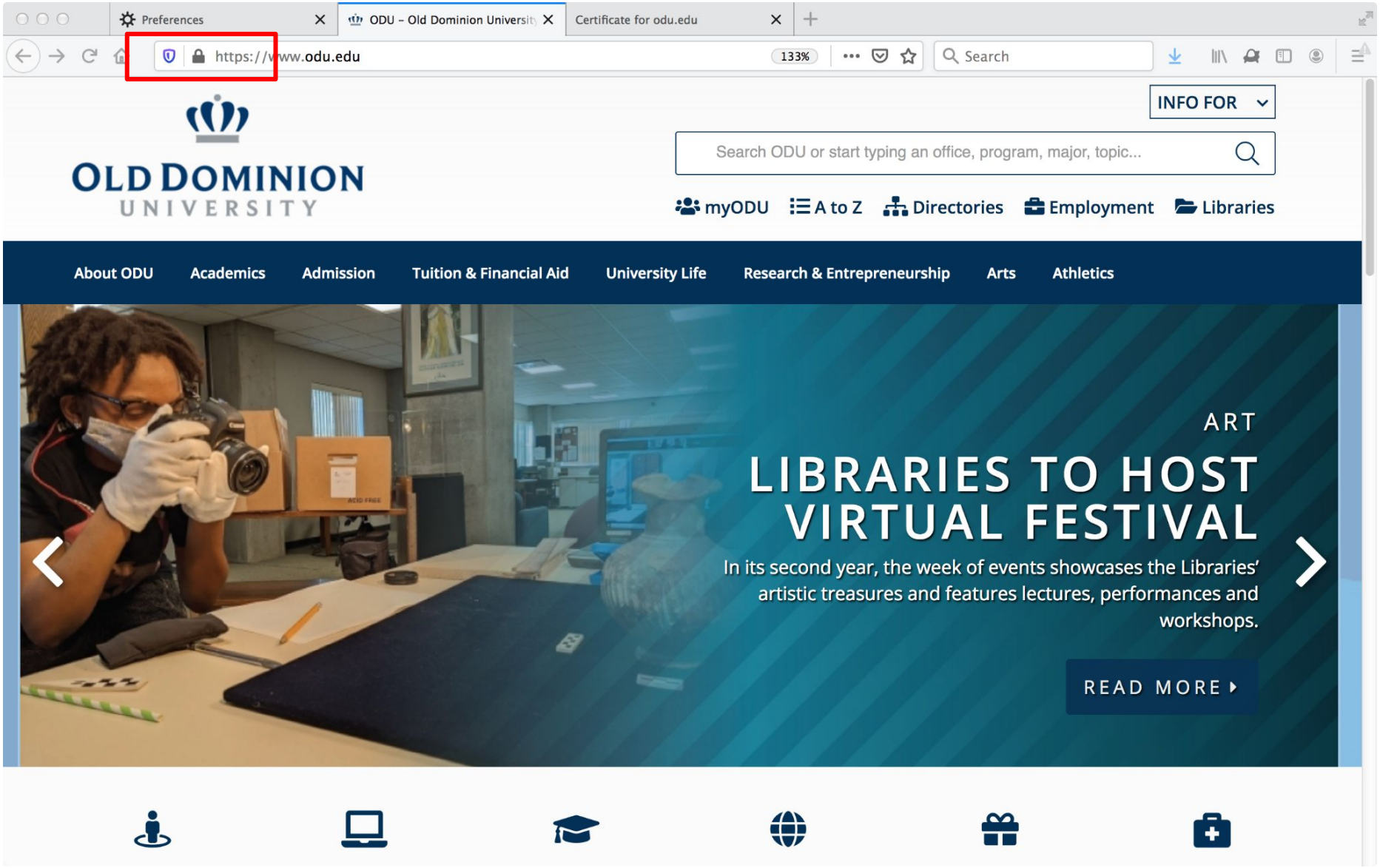

Privacy error
(2) Michael L. Nelson (@phon
Michael
Not Secure
https://www.odu.edu
Your connection is not private
Attackers might be trying to steal your information from www.odu.edu (for example, passwords, messages, or credit cards). Learn more
NET::ERR_CERT_AUTHORITY_INVALID
Automatically send some system information and page content to Google to help detect dangerous apps and sites. Privacy policy
ADVANCED
Back to safety

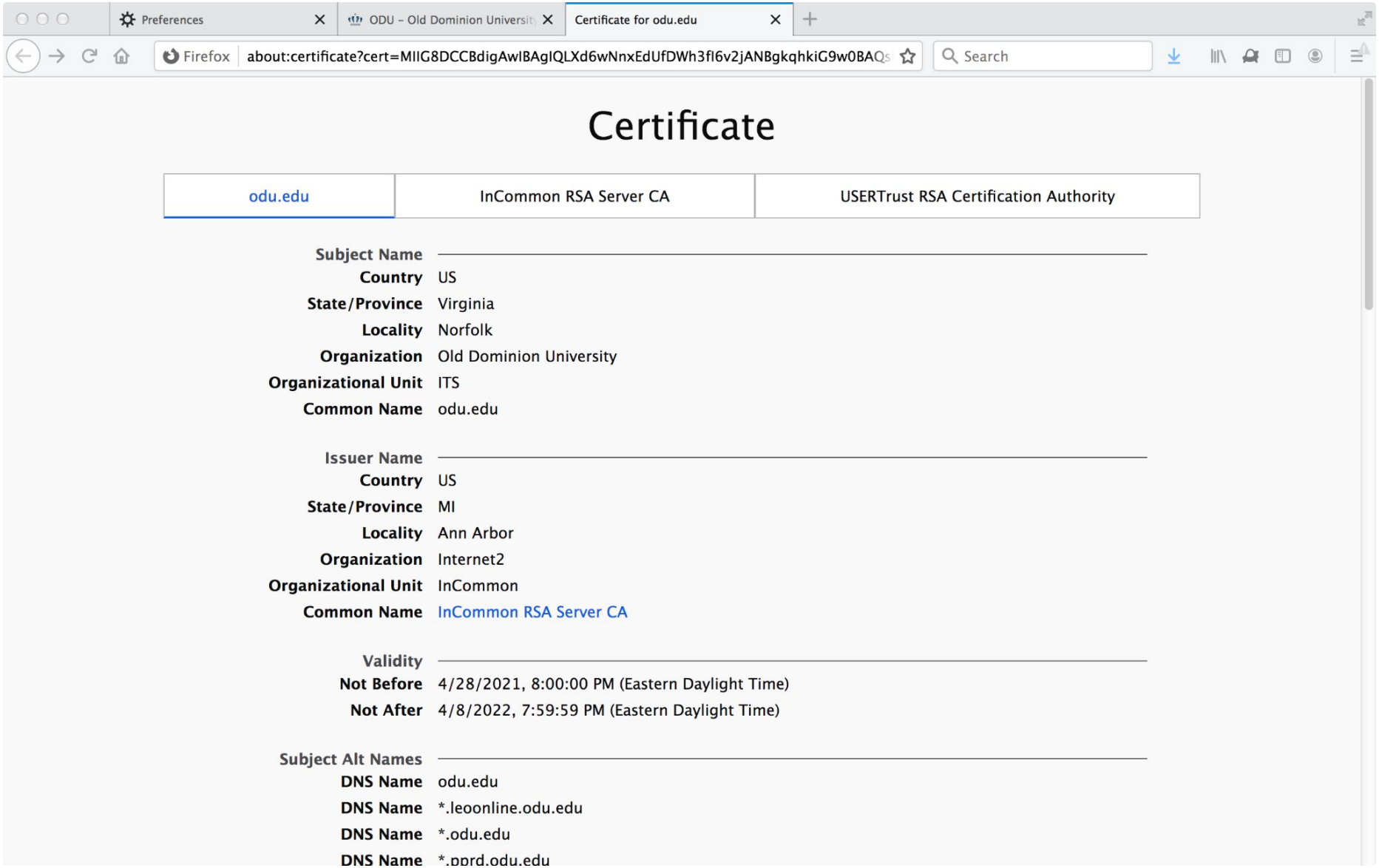

Privacy error
(2) Michael L. Nelson (@phone
Michael
Not Secure
https://www.odu.edu
USERTrust RSA Certification Authority
InCommon RSA Server CA
odu.edu
odu.edu
Issued by: InCommon RSA Server CA
Expires: Friday, April 8, 2022 at 7:59:59 PM Eastern Daylight Time
This certificate was signed by an untrusted issuer
Details
Subject Name
Country US
State/Province Virginia
Locality Norfolk
Organization Old Dominion University
Organizational Unit ITS
Common Name odu.edu
Issuer Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA
OK


Privacy error
(2) Michael L. Nelson (@phone
Michael
Not Secure
https://www.odu.edu
USERTrust RSA Certification Authority
InCommon RSA Server CA
odu.edu
odu.edu
Issued by: InCommon RSA Server CA
Expires: Friday, April 8, 2022 at 7:59:59 PM Eastern Daylight Time
This certificate was signed by an untrusted issuer
Details
Keychain Access
Click to unlock the System Roots keychain.
Keychains
login
Micr...ertificates
Local Items
System
System Roots
Category
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
DigiCert Global Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 7:00:00 PM Eastern Standard Time
This certificate is marked as trusted for all users
Name
Kind
Expires
Keychain
TRUST2408 OCES Primary CA
Trusted Certificate Services
Trustis FPS Root CA
TWCA Global Root CA
TWCA Root C...fication Authority
UCA Global Root certificate Dec 30, 2037, 7:00:00 PM System Roots
UCA Root certificate Dec 30, 2029, 7:00:00 PM System Roots
VeriSign Clas...on Authority - G3 certificate Jul 16, 2036, 7:59:59 PM System Roots
VeriSign Clas...on Authority - G3 certificate Jul 16, 2036, 7:59:59 PM System Roots
VeriSign Clas...on Authority - G3 certificate Jul 16, 2036, 7:59:59 PM System Roots
VeriSign Clas...on Authority - G4 certificate Jan 18, 2038, 6:59:59 PM System Roots
VeriSign Clas...on Authority - G5 certificate Jul 16, 2036, 7:59:59 PM System Roots
VeriSign Clas...on Authority - G3 certificate Jul 16, 2036, 7:59:59 PM System Roots
VeriSign Univ...fication Authority certificate Dec 1, 2037, 6:59:59 PM System Roots
Visa eCommerce Root certificate Jun 23, 2022, 8:16:12 PM System Roots
Visa Informa...Delivery Root CA certificate Jun 29, 2025, 1:42:42 PM System Roots
VRK Gov. Root CA certificate Dec 18, 2023, 8:51:08 AM System Roots
WellsSecure...ertificate Authority certificate Dec 13, 2022, 7:07:54 PM System Roots
XRamp Glob...ification Authority certificate Jan 1, 2035, 12:37:19 AM System Roots
Copy
155 items
No "USERTrust RSA Certification Authority" in the list

Firefox has its own CA list, all other popular browsers use the system CA list

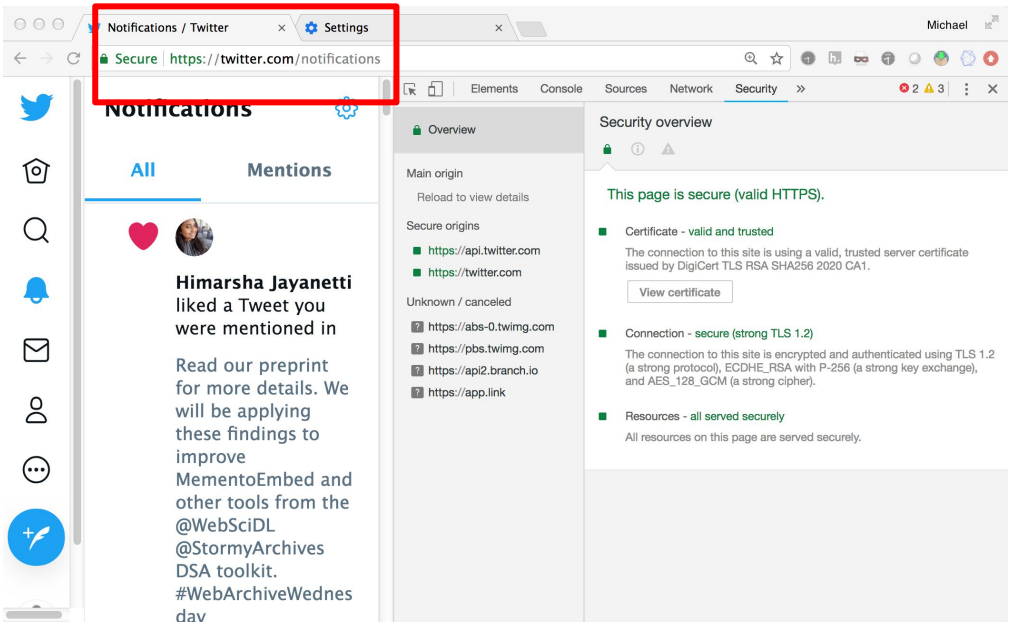

Preferences
ODU – Old Dominion University
Certificate for odu.edu
Firefox
about:preferences#privacy
Search
Find in Preferences
General
Home
Search
Privacy & Security
Sync
Allow Firefox to send technical and interaction data to Mozilla Learn more
Allow Firefox to make personalized extension recommendations Learn more
Certificate Manager
Your Certificates
Authentication Decisions
People
Servers
Authorities
You have certificates on file that identify these certificate authorities
Certificate Name
Security Device
USERTrust RSA Certification Authority
Builtin Object Token
USERTrust RSA Domain Validation Secure S...
Software Security Device
TrustAsia RSA DV SSL Server CA
Software Security Device
WebSpace-Forum Server CA
Software Security Device
View...
Edit Trust...
Import...
Export...
Delete or Distrust...
OK
Query OCSP responder servers to confirm the current validity of certificates
View Certificates...
Security Devices...
Extensions & Themes
Firefox Support


Preferences
ODU – Old Dominion University
Certificate for odu.edu
https://www.odu.edu
133%
Search
INFO FOR
OLD DOMINION UNIVERSITY
Search ODU or start typing an office, program, major, topic...
myODU
A to Z
Directories
Employment
Libraries
About ODU
Academics
Admission
Tuition & Financial Aid
University Life
Research & Entrepreneurship
Arts
Athletics
ART
LIBRARIES TO HOST VIRTUAL FESTIVAL
In its second year, the week of events showcases the Libraries' artistic treasures and features lectures, performances and workshops.
READ MORE


Preferences
ODU – Old Dominion University
Certificate for odu.edu
Firefox
about:certificate?cert=MIIG8DCCBdigAwIBAgIQLXd6wNnxEdUfDWh3fI6v2jANBgkqhkiG9w0BAQs
Search
Certificate
odu.edu
InCommon RSA Server CA
USERTrust RSA Certification Authority
Subject Name
Country US
State/Province Virginia
Locality Norfolk
Organization Old Dominion University
Organizational Unit ITS
Common Name odu.edu
Issuer Name
Country US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA
Validity
Not Before 4/28/2021, 8:00:00 PM (Eastern Daylight Time)
Not After 4/8/2022, 7:59:59 PM (Eastern Daylight Time)
Subject Alt Names
DNS Name odu.edu
DNS Name *.leoonline.odu.edu
DNS Name *.odu.edu
DNS Name *.pprd.odu.edu

# HTTPS requirements for lock icon

- All elements on the page must be fetched using HTTPS
- For all elements
  - HTTPS certificate must be issued by a CA trusted by browser
  - HTTPS certificate must not be expired
  - HTTPS certificate CommonName or SubjectAlternativeName must match the URL



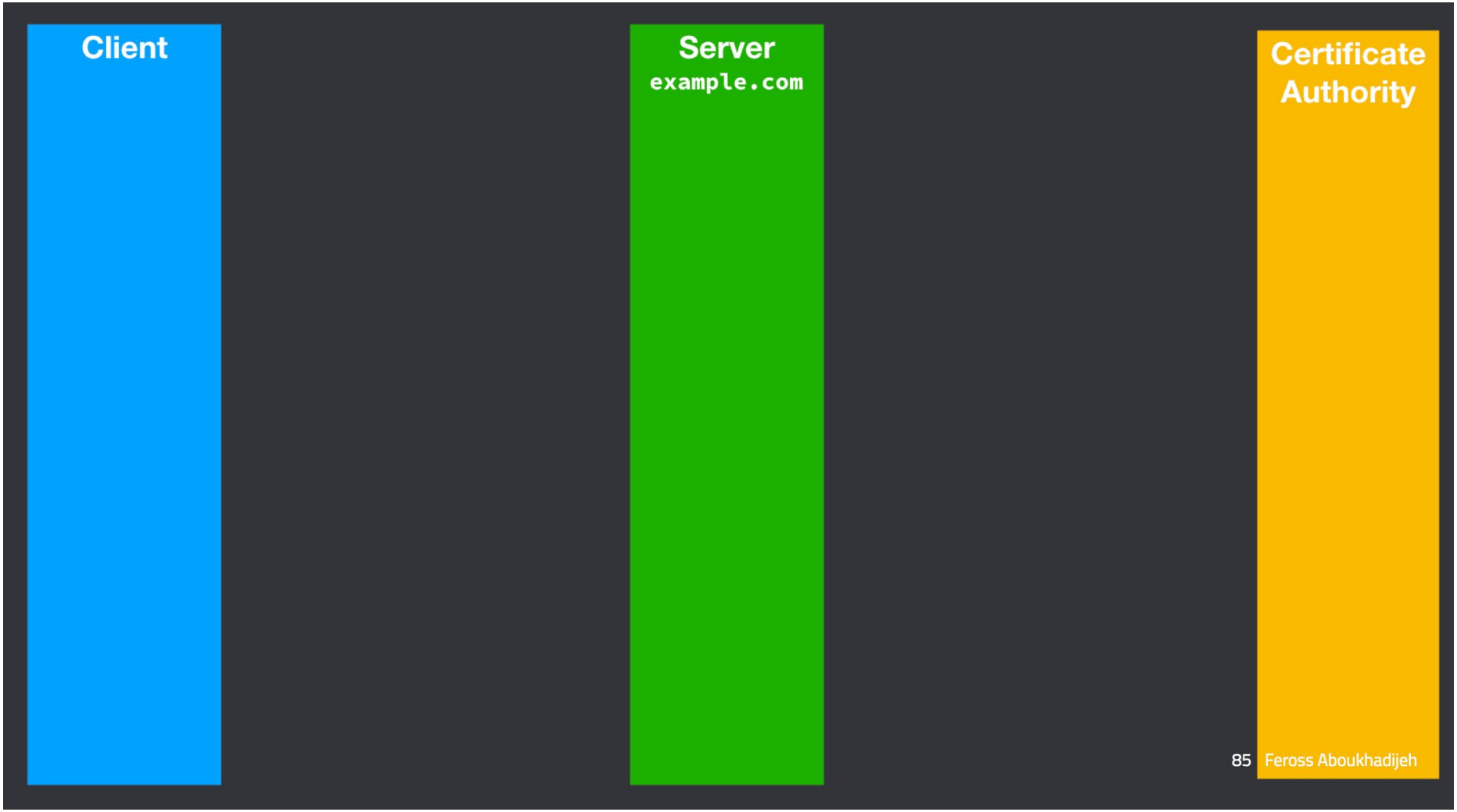
# Transitivity of trust: “A friend of mine” vs. “A friend of ours”

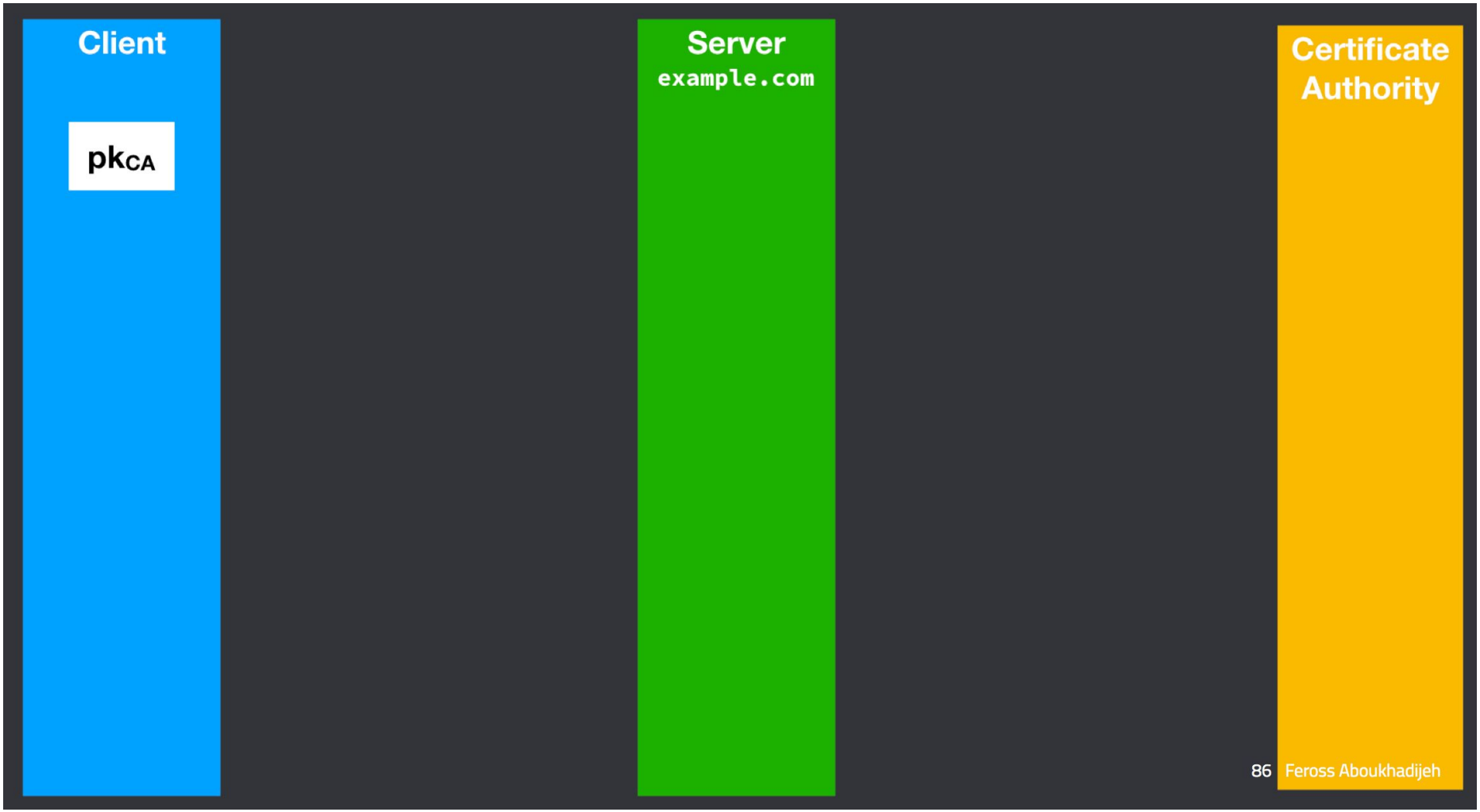
# Certificate exchange

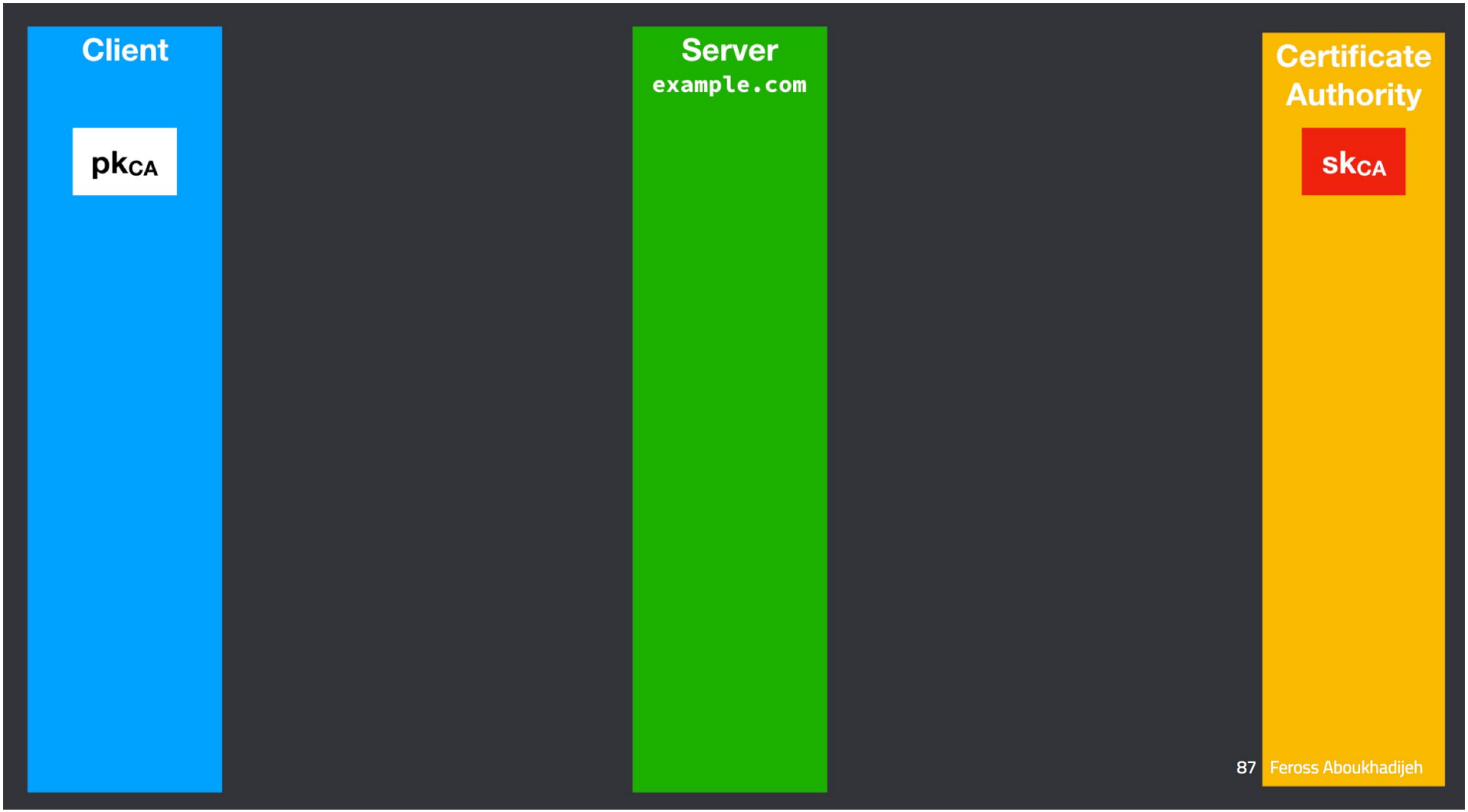
Client

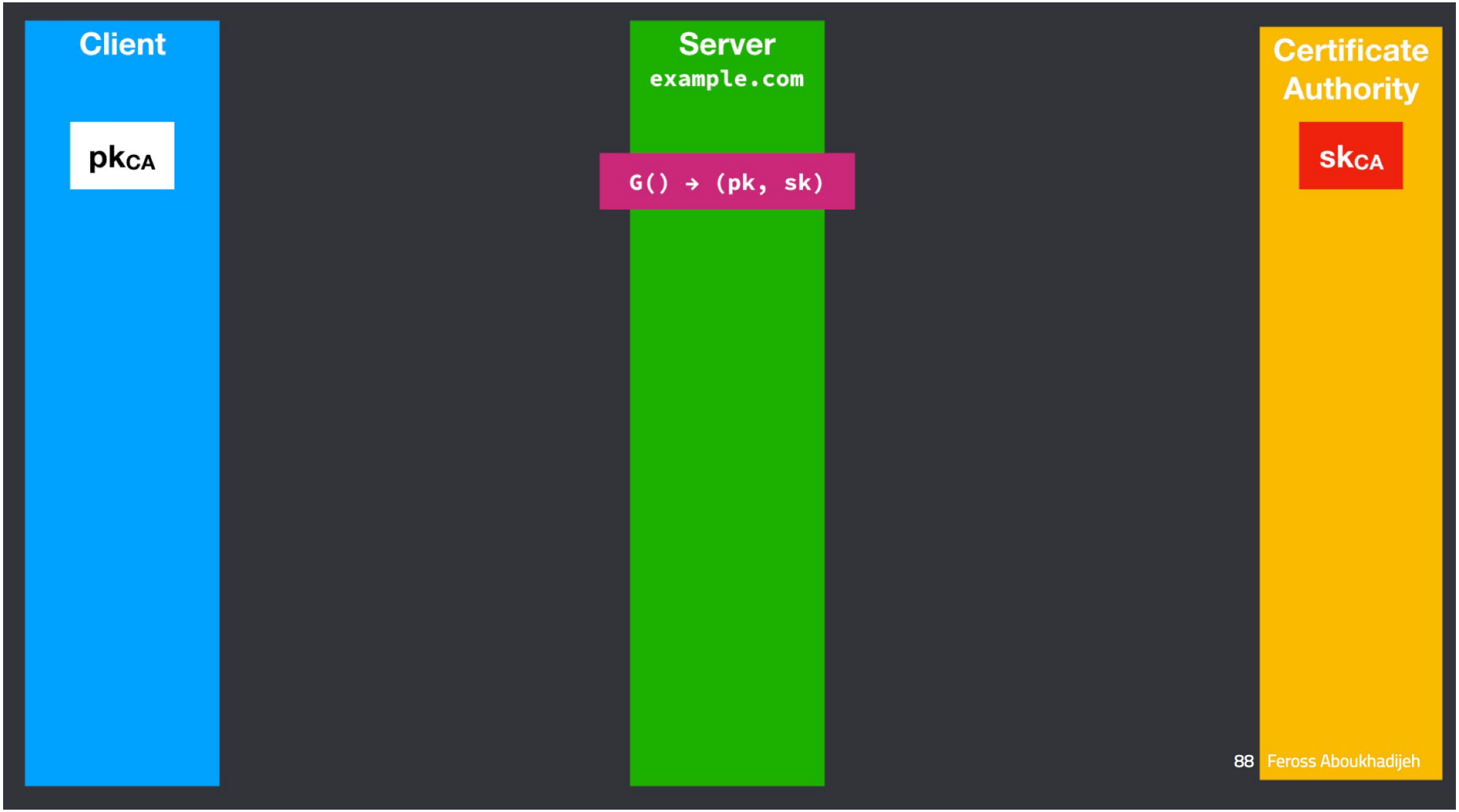
Server

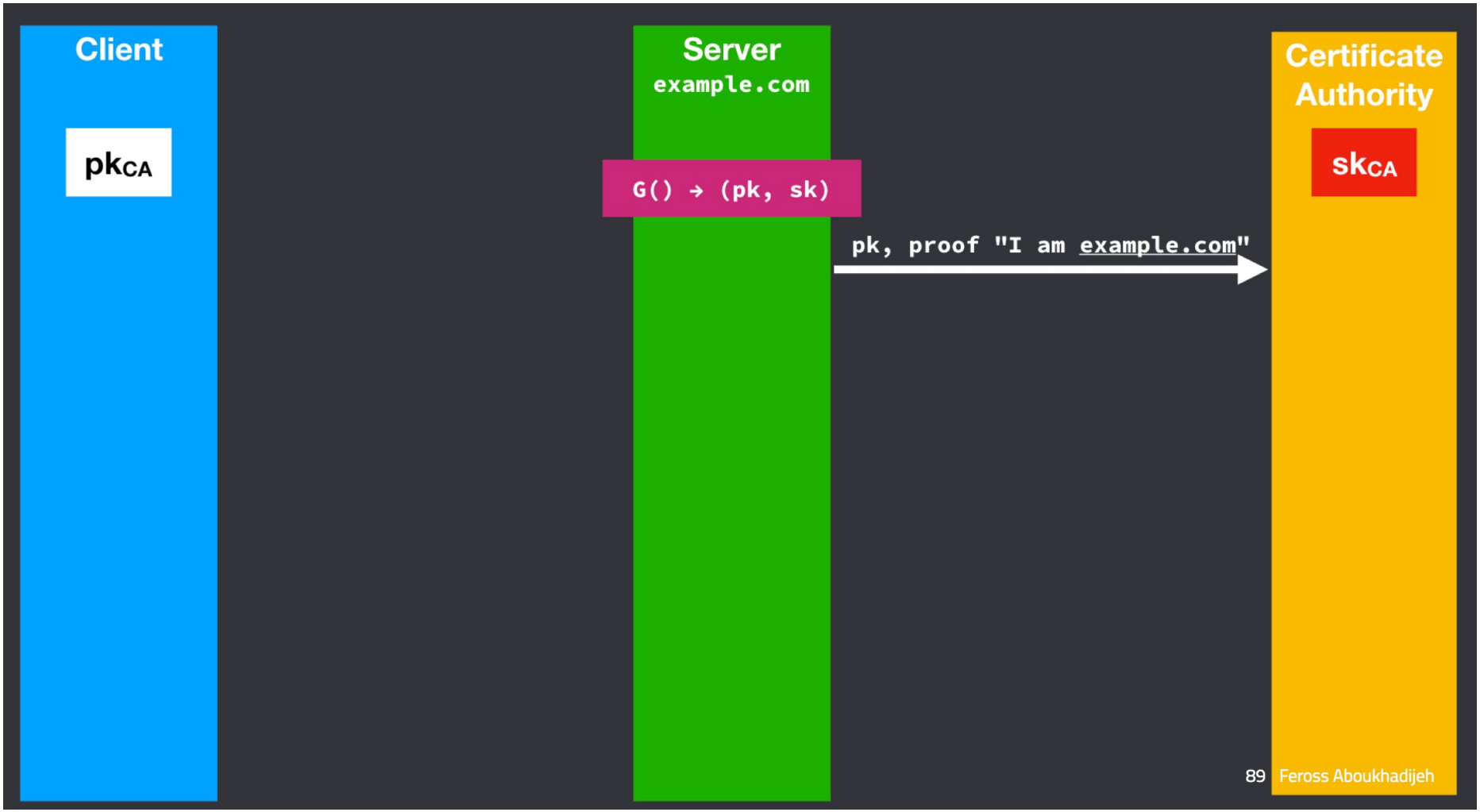
Certificate Authority

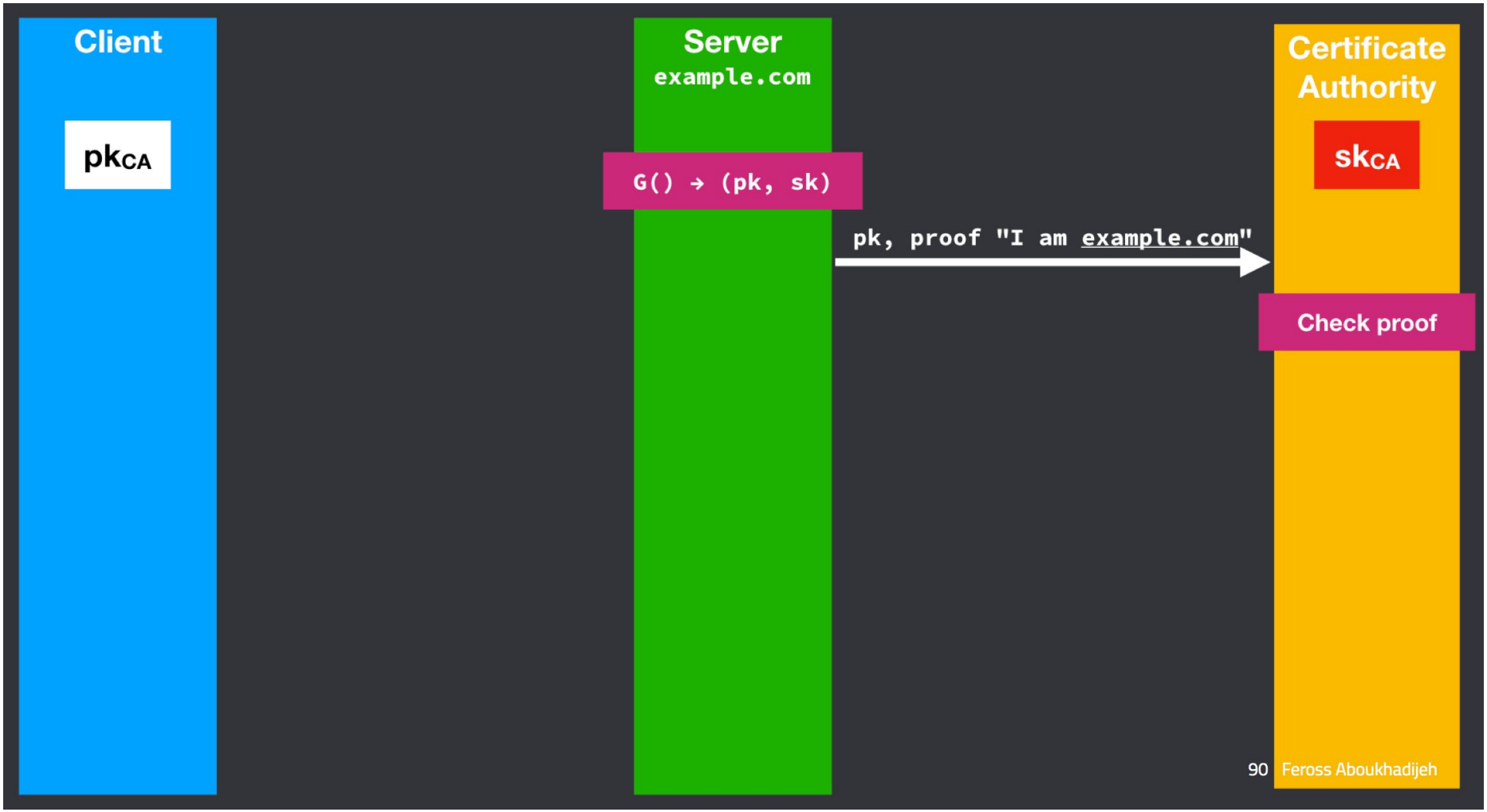

Client
Server
example.com
Certificate Authority
85
Feross Aboukhadijeh

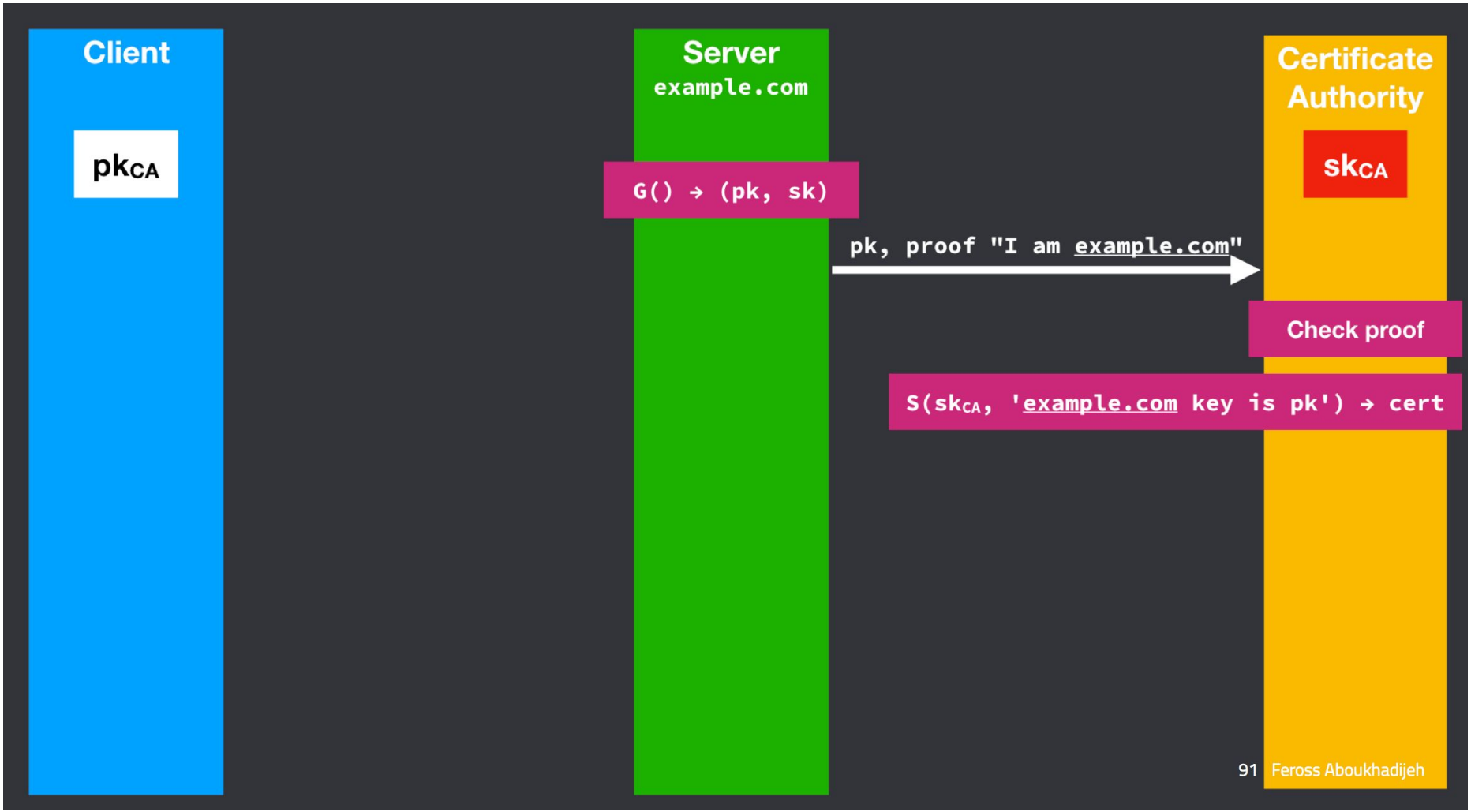

Client
pkCA
Server
example.com
Certificate Authority
86
Feross Aboukhadijeh

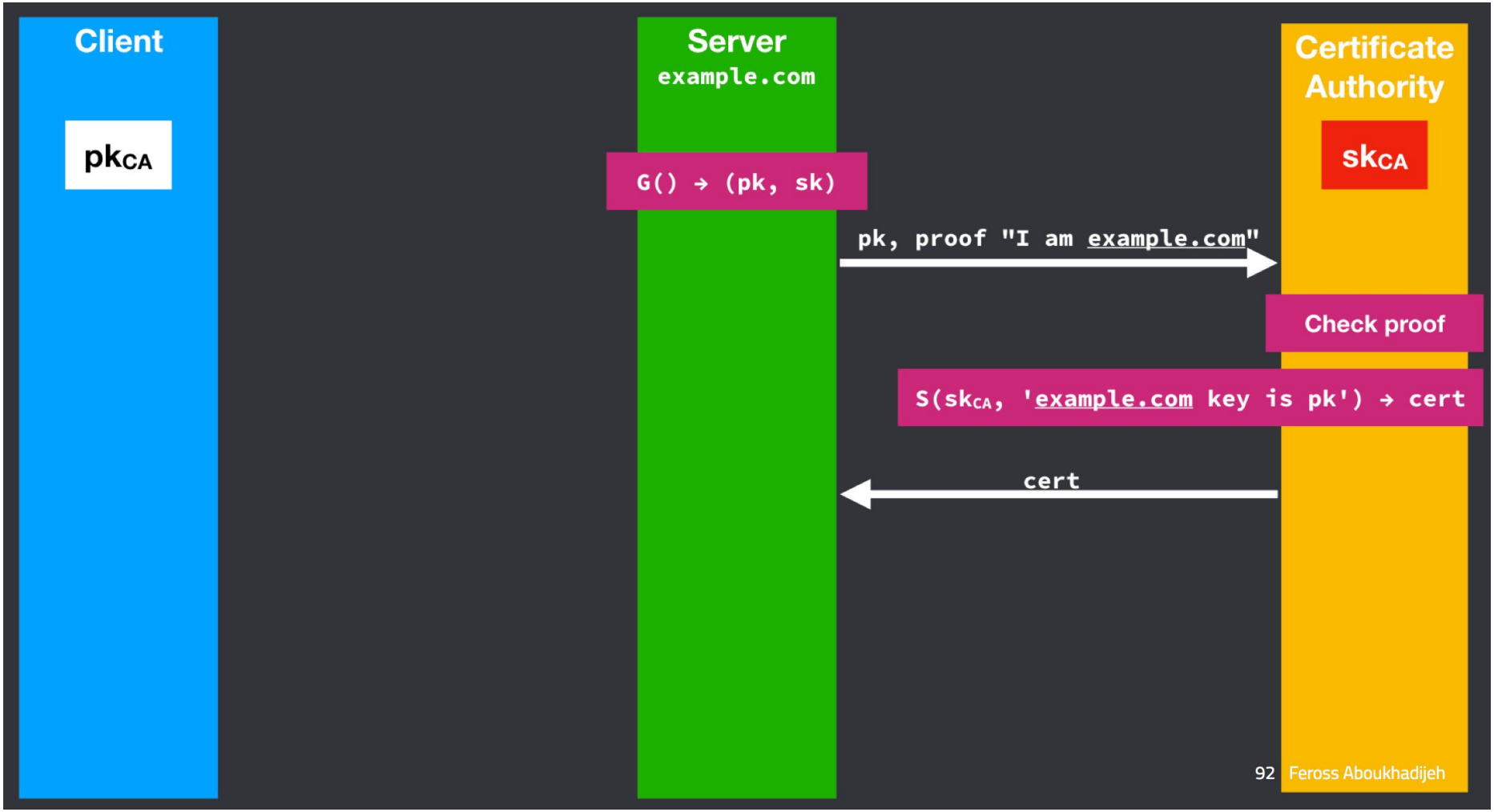

Client
pkCA
Server
example.com
Certificate Authority
skCA
87
Feross Aboukhadijeh

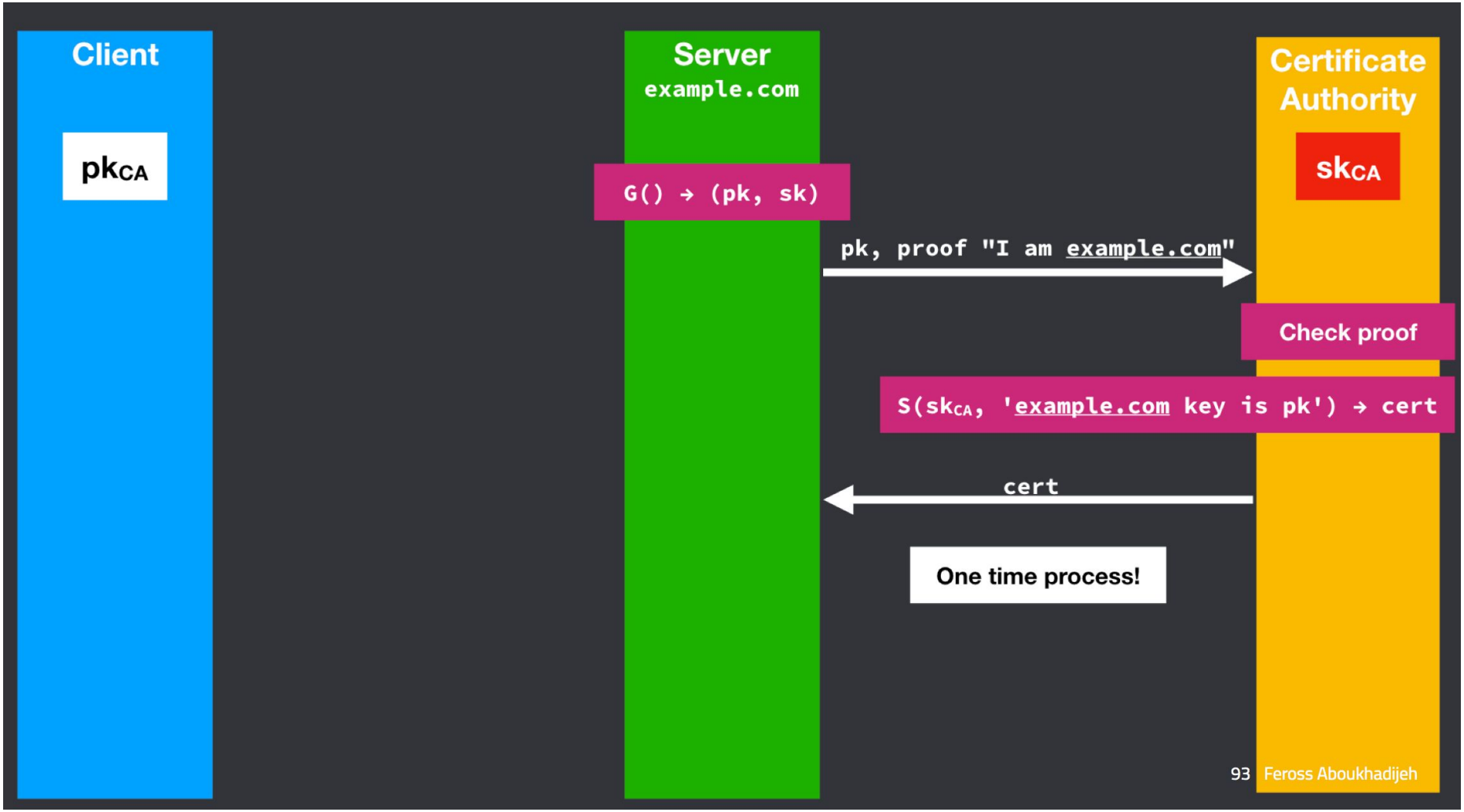

Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
88
Feross Aboukhadijeh

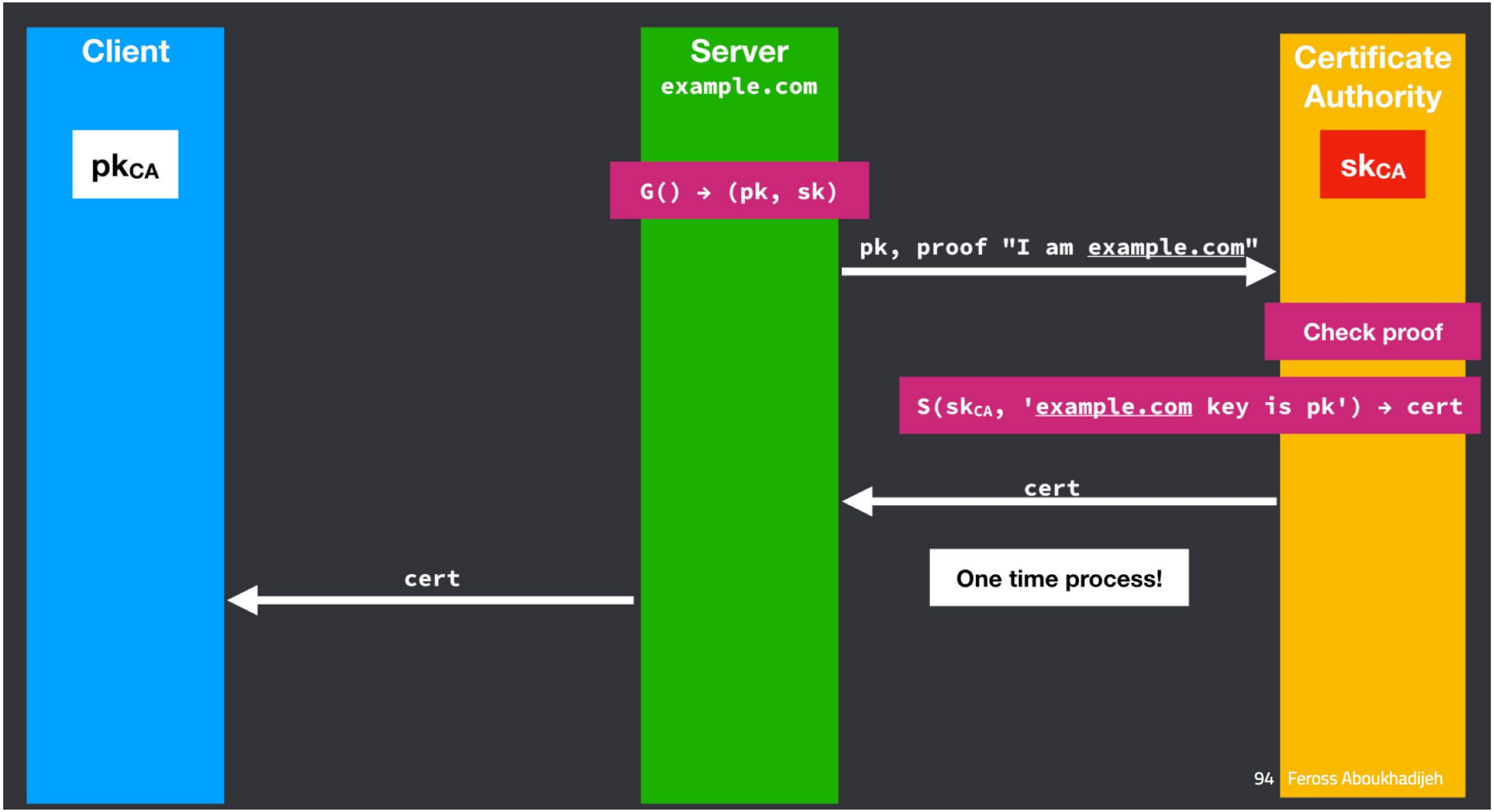

Client
pkCA
Server
example.com
G() → (pk, sk)
pk, proof "I am example.com"
Certificate Authority
skCA
89
Feross Aboukhadijeh

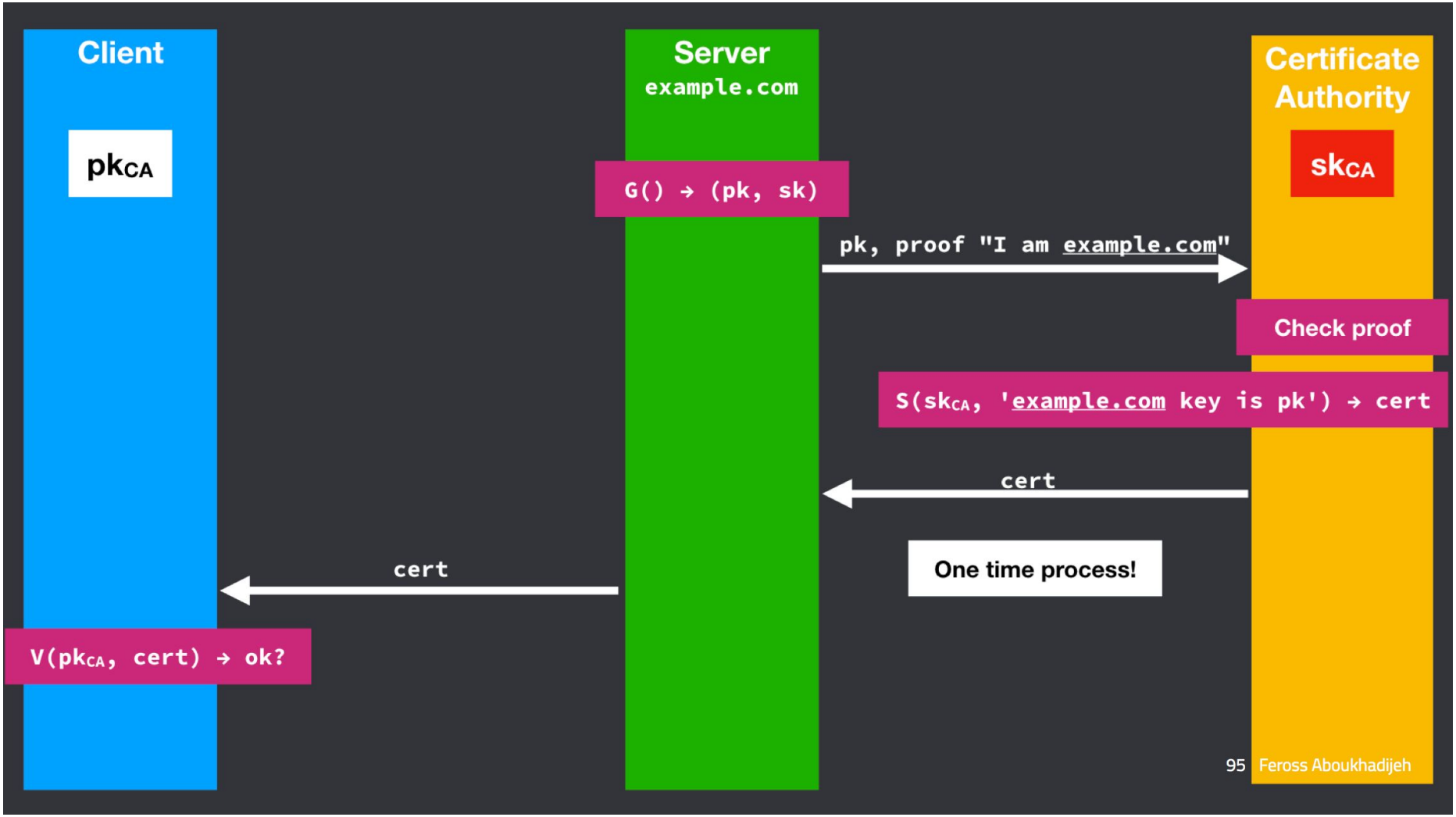

Client
pkCA
Server
example.com
G() → (pk, sk)
pk, proof "I am example.com"
Certificate Authority
skCA
Check proof
90
Feross Aboukhadijeh

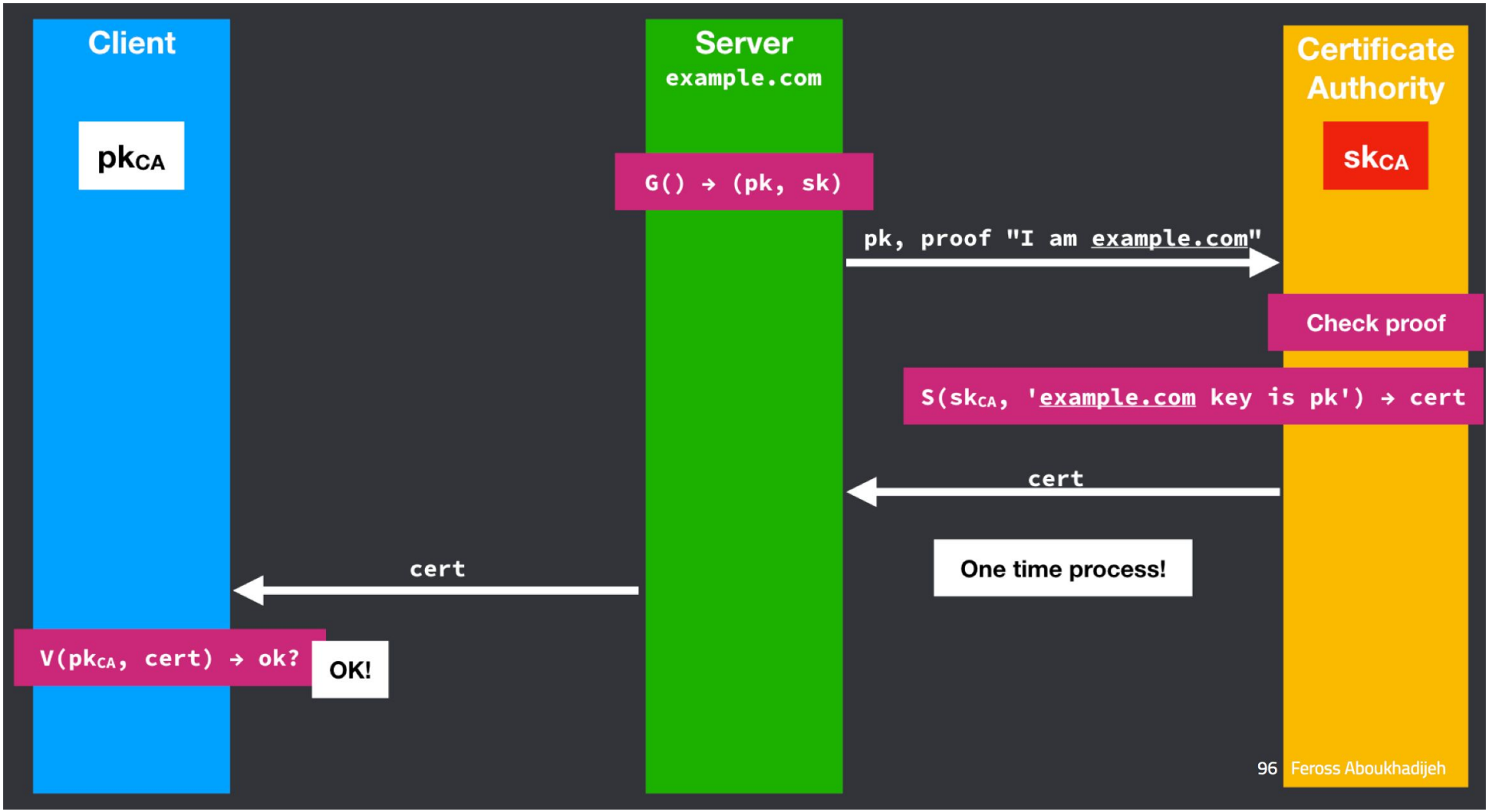

Client
pkCA
Server
example.com
G() → (pk, sk)
pk, proof "I am example.com"
Certificate Authority
skCA
Check proof
S(skCA, 'example.com key is pk') → cert
91
Feross Aboukhadijeh

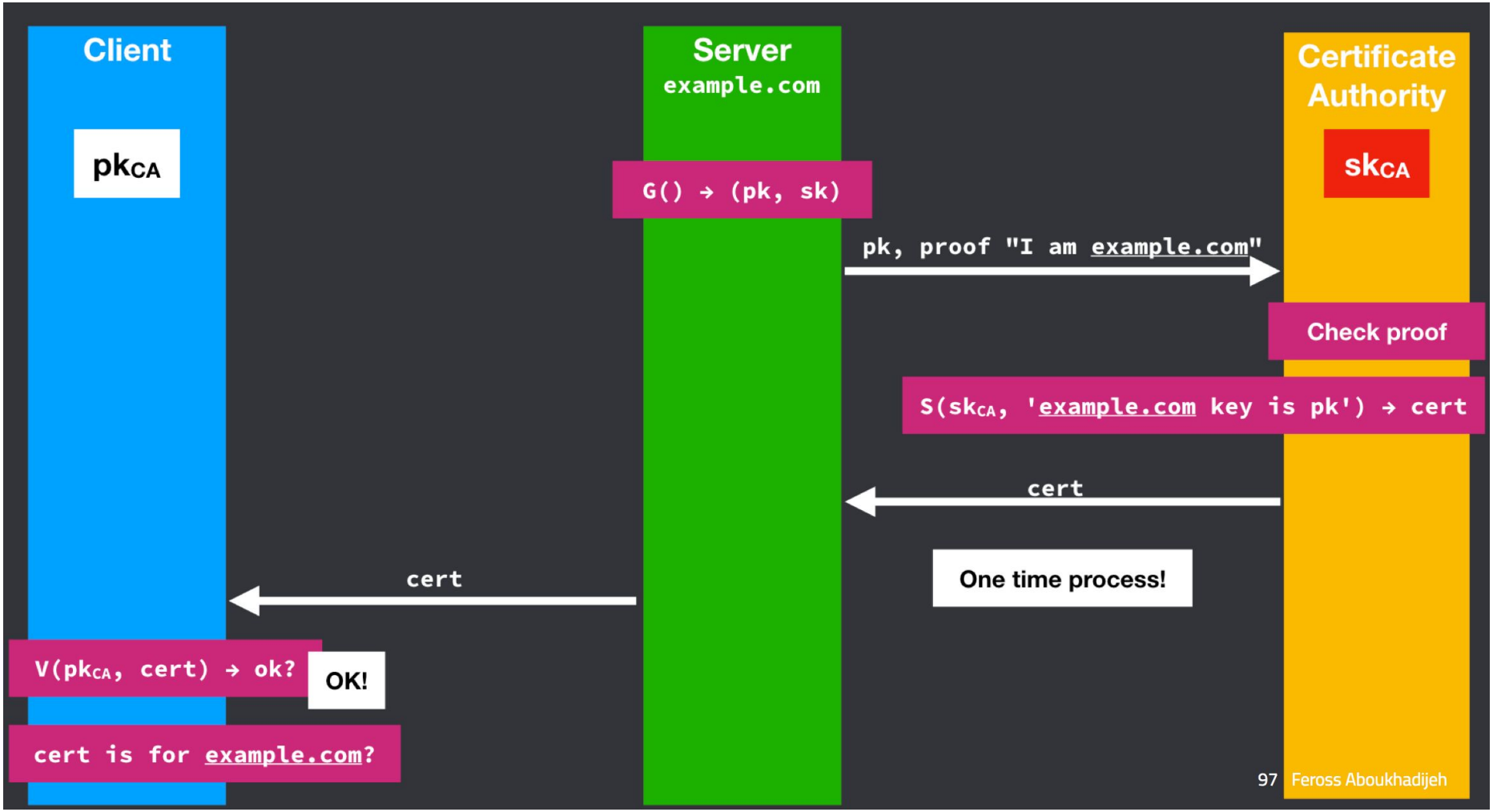

Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
92
Feross Aboukhadijeh

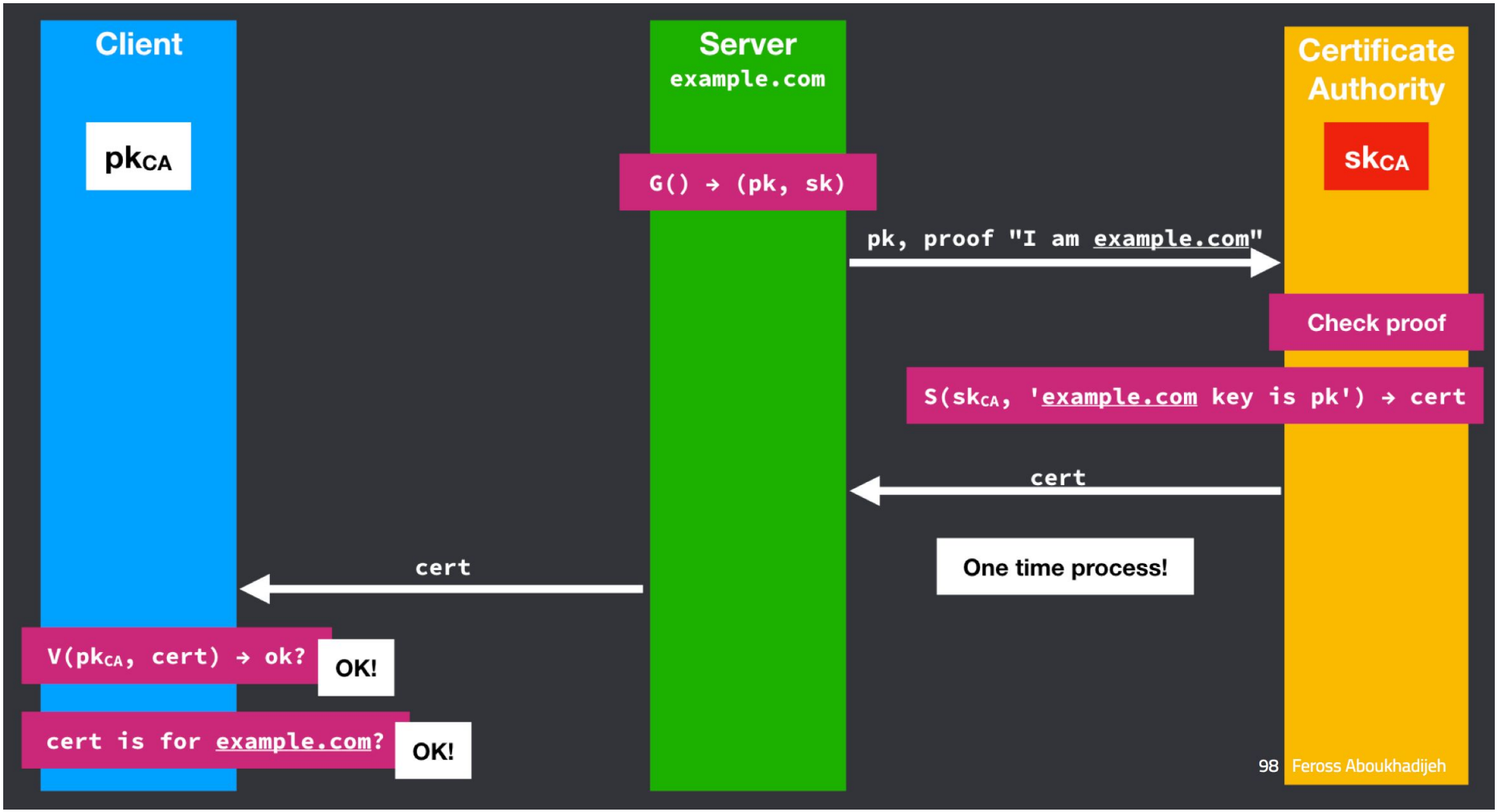

Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
93
Feross Aboukhadijeh

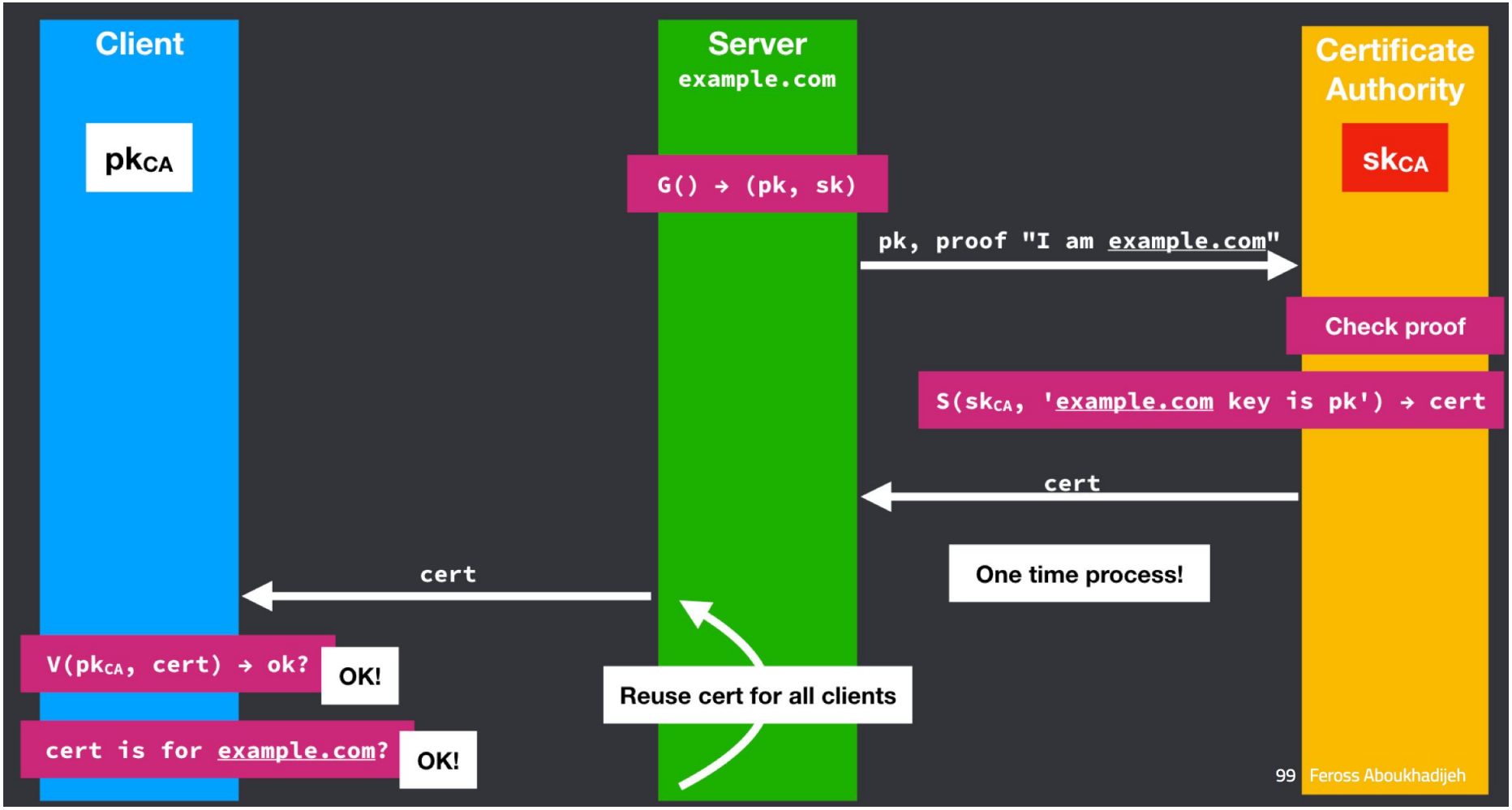

Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
cert
94 Feross Aboukhadijeh


Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
cert
V(pkCA, cert) → ok?
95
Feross Aboukhadijeh


Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
cert
V(pkCA, cert) → ok?
OK!
96 Feross Aboukhadijeh


Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
cert
V(pkCA, cert) → ok?
OK!
cert is for example.com?
97 Feross Aboukhadijeh

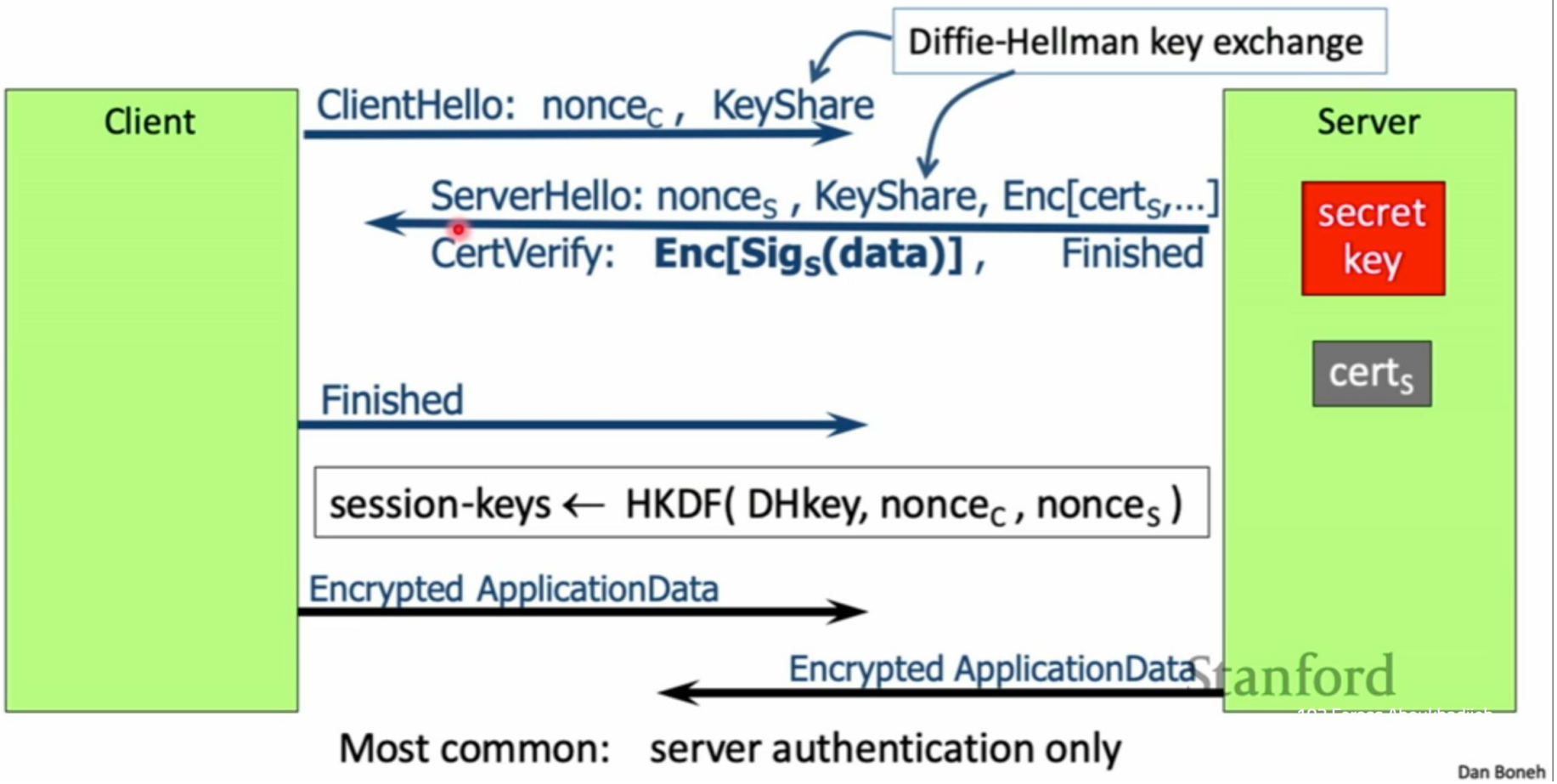

Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
cert
V(pkCA, cert) → ok?
OK!
cert is for example.com?
OK!
98 Feross Aboukhadijeh


Client
pkCA
Server
example.com
G() → (pk, sk)
Certificate Authority
skCA
pk, proof "I am example.com"
Check proof
S(skCA, 'example.com key is pk') → cert
cert
One time process!
cert
V(pkCA, cert) → ok?
OK!
Reuse cert for all clients
cert is for example.com?
OK!
99
Feross Aboukhadijeh

# TLS 1.3

- TLS 1.3 is the latest version of TLS which replaces TLS 1.2, which replaced TLS 1.1, 1.0, SSL 3.0, 2.0, 1.0.
- Goal: "provide privacy and reliability between two communicating applications"
- Two phase protocol
  - Handshake protocol: Establish a shared secret key using public-key cryptography
  - Record protocol: Transmit data using the negotiated key

# TLS 1.3 session setup (simplified)



# TLS 1.3 properties

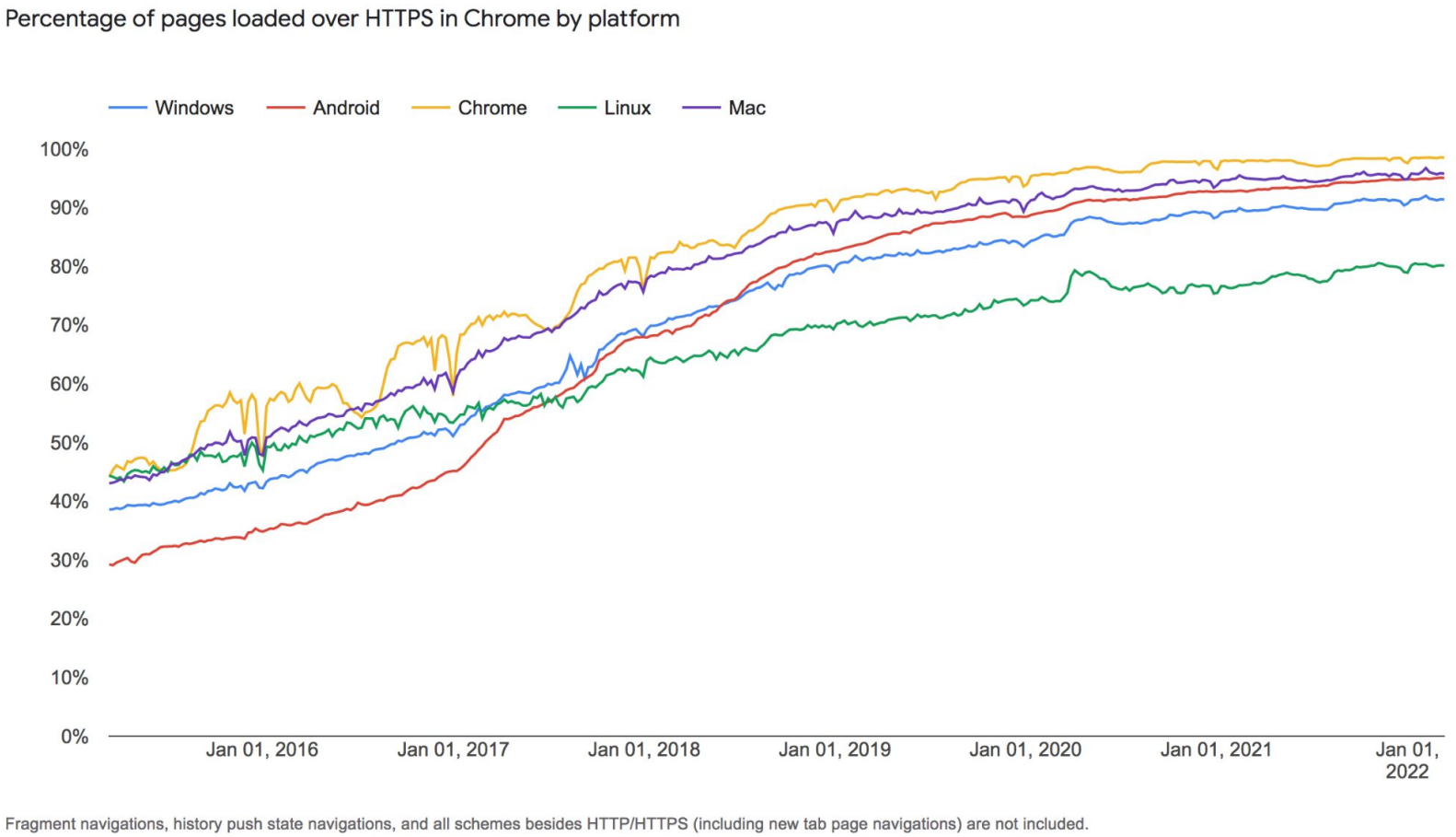
- Nonces: Prevent replay of an old session
- Forward secrecy: server compromise does not expose old sessions
- Some identity protection: certificates are sent encrypted
- One-sided authentication: Client authenticates the server using the server's certificate
  - TLS has support for mutual authentication ("client certificates") but it is rarely used

# HTTPS adoption

- Survey of top 100 non-Google sites on the Internet, which account for 25% of website traffic worldwide
- Sites that default to HTTPS
  - 96 / 100 (2019)
  - 97 / 100 (2022)
- • Sites that work on HTTPS
  - 90 / 100 (2019)
  - 100 / 100 (2022)

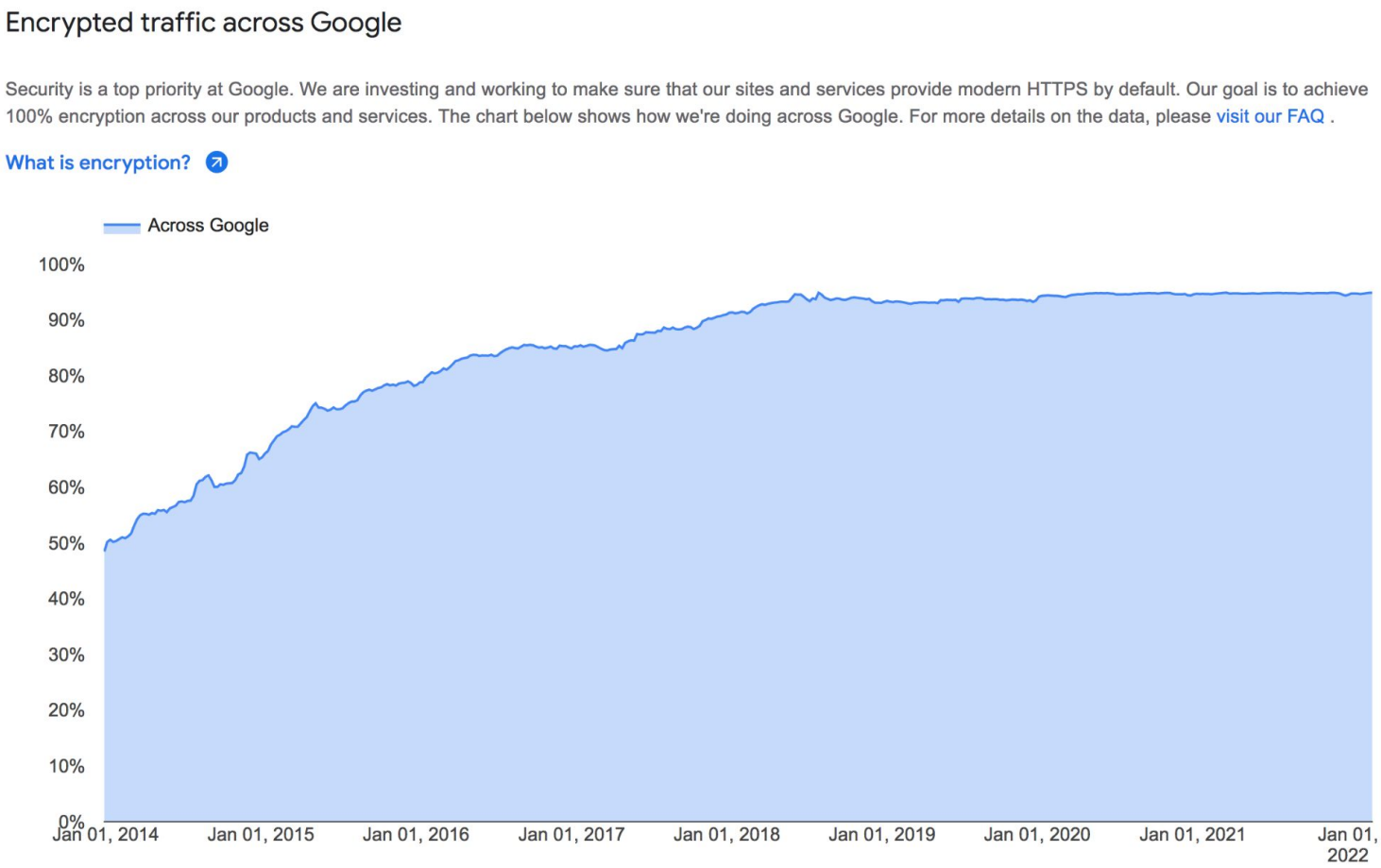
https://transparencyreport.google.com/https/overview?hl=en

# Percentage of pages loaded over HTTPS in Chrome by platform




https://transparencyreport.google.com/https/overview?hl=en

# Encrypted traffic across Google sites

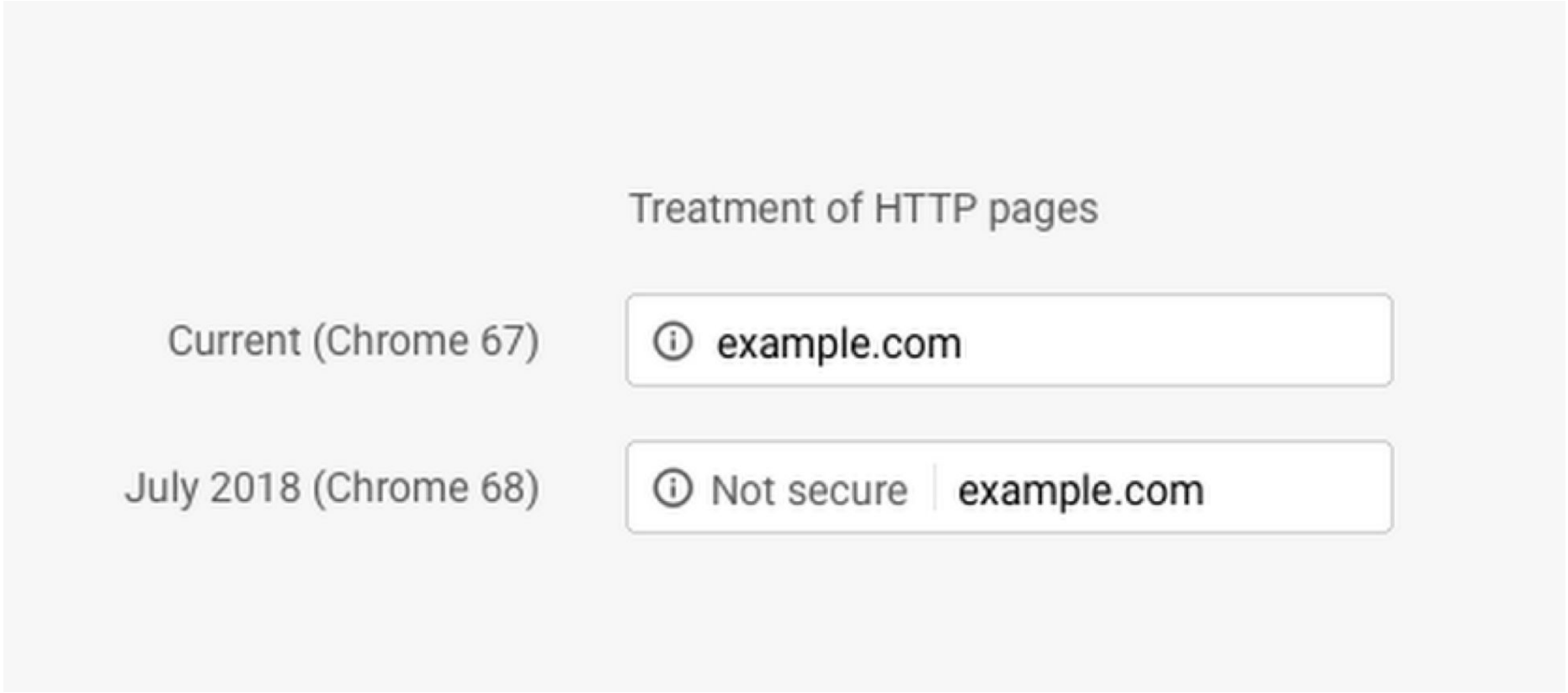



https://transparencyreport.google.com/https/overview?hl=en

# Why not 100%
## (it's been stuck at ~95% since ~2018)

- Old excuses (not true anymore)
  - Crypto is slow
  - Ad networks do not support HTTPS
- "Mobile devices account for the vast majority of unencrypted end user traffic that originates from a given set of surveyed Google services. Some older devices cannot support modern encryption, standards, or protocols. Unfortunately, these devices may no longer support software updates and, as a result, may never support encryption"

# “Secure” is now the assumed default, with “not secure” the alert message



# TLS certificate chains

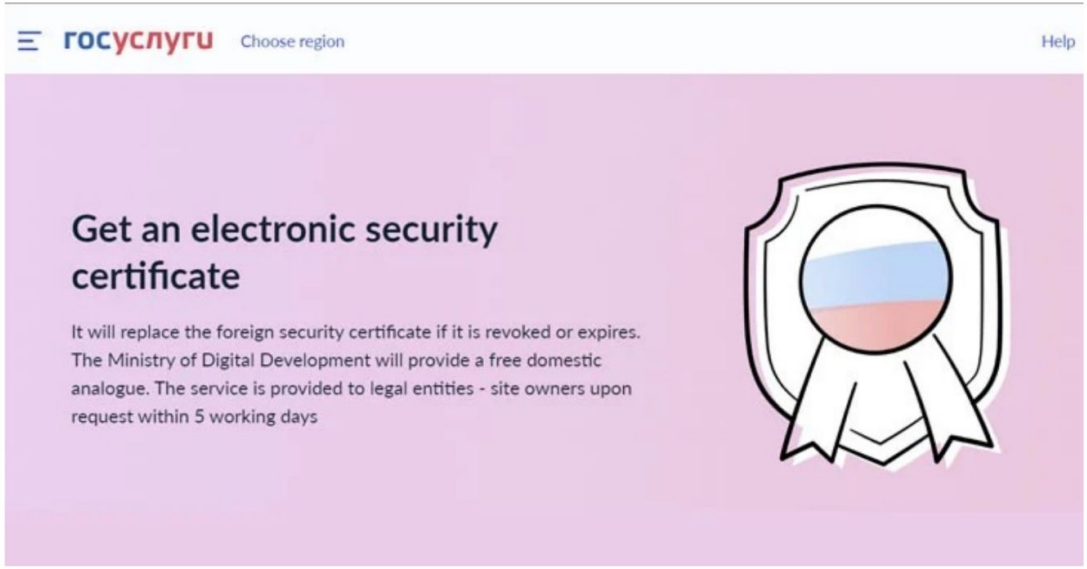
- How many CAs are there?
  - Top-level CAs = ~60
  - Intermediate CAs = ~1200
- If any single CA is compromised, security of all websites on the Internet could be compromised – yikes!

Russia Issues Its Own TLS Certs

https://threatpost.com/russia-issues-its-own-tls-certs/178891/

threatpost Podcasts / Malware / Vulnerabilities / InfoSec Insiders / Webinars Search

# Russia Issues Its Own TLS Certs




Author:
Lisa Vaas

March 11, 2022 / 1:34 pm

2 minute read

The country's citizens are being blocked from the internet because foreign certificate authorities can't accept payments due to Ukraine-related sanctions, so it created its own CA.

Russia is offering its own trusted Transport Layer Security (TLS) certificate authority (CA) to replace certificates that need to be renewed by foreign countries. As it is, a pile of sanctions imposed in the wake of Russia's invasion of Ukraine is gumming up its citizen's access to websites.

INFOSEC INSIDER

**The Uncertain Future of IT Automation**
March 8, 2022

**6 Cyber-Defense Steps to Take Now to Protect Your Company**
February 25, 2022 1

**The Harsh Truths of Cybersecurity in 2022, Part II**
February 24, 2022 2

**3 Tips for Facing the Harsh Truths of Cybersecurity in 2022, Part I**
February 9, 2022

**'Long Live Log4Shell': CVE-2021-44228 Not Dead Yet**
February 4, 2022

https://threatpost.com/russia-issues-its-own-tls-certs/178891/

NEWS

# Hackers spied on 300,000 Iranians using fake Google certificate

## Investigation reveals month-long, massive Gmail snooping campaign

By Gregg Keizer
Senior Reporter, Computerworld | SEP 6, 2011 5:43 AM PST

About 300,000 Iranians had their Gmail accounts compromised and their messages read by hackers, according to a forensics firm that has investigated the theft of hundreds of digital certificates from a Dutch company.

Although the report did not identify the hacker, or hackers, who may have spied on the Iranian users, security researchers have pointed to Iran's government, which has been linked to other attempts to intercept the communications of activists and protesters.

https://www.computerworld.com/article/2510951/hackers-spied-on-300-000-iranians-using-fake-google-certificate.html

SUBSCRIBE | ABOUT | RSS

cyberscoop

SNG SCOOP NEWS GROUP

GOVERNMENT | TRANSPORTATION | HEALTHCARE | TECHNOLOGY | FINANCIAL | WATCH | LISTEN | ATTEND | CONTENT

TECHNOLOGY

# Trustico revokes 23,000 SSL certificates due to compromise

https://www.cyberscoop.com/trustico-digicert-ssl-certificates-revoked/

The Register®
Biting the hand that feeds IT

Security

# New hack on Comodo reseller exposes private data

## And then there were four

By Dan Goodin 24 May 2011 at 19:58 5 SHARE ▼

Yet another official reseller of SSL certificate authority Comodo has suffered a security breach that allowed attackers to gain unauthorized access to data.

Brazil-based ComodoBR is at least the fourth Comodo partner to be compromised this year. In March, the servers of a separate registration authority were hacked by attackers who used their access to forge counterfeit certificates signed with Comodo's root signing key. Comodo admitted that two more of its resellers were hit in similar attacks, although no keys were issued.

Comodo has so far declined to name the resellers.

The SQL-injection attack on ComodoBR exploited vulnerabilities in the company's web applications that allowed the hackers to pass database commands to the website's backend server. The attackers posted two data files that appeared to show information related to certificate signing

https://www.theregister.com/2011/05/24/comodo_reseller_hacked/

# Comodo reseller hack

- The attackers registered fraudulent certificates for gmail.com, google.com, login.yahoo.com, login.skype.com, addons.mozilla.com, and login.live.com
- Quote from Comodo president and CEO
  - "So as a summary: its an SQL attack (fairly common) on a company in Brazil who sells some of our products.” he wrote in an email. “Nothing to report really."

mozilla

# Mozilla Security Blog

MAR 12 2018

## Distrust of Symantec TLS Certificates

Kathleen Wilson

A Certification Authority (CA) is an organization that browser vendors (like Mozilla) trust to issue certificates to websites. Last year, Mozilla published and discussed a set of issues with one of the oldest and largest CAs run by Symantec. The discussion resulted in the adoption of a consensus proposal to gradually remove trust in all Symantec TLS/SSL certificates from Firefox. The proposal includes a number of phases designed to minimize the impact of the change to Firefox users:

- January 2018 (Firefox 58): Notices in the Browser Console warn about Symantec certificates issued before 2016-06-01, to encourage site owners to replace their TLS certificates.
- May 2018 (Firefox 60): Websites will show an untrusted connection error if they use a TLS certificate issued before 2016-06-01 that chains up to a Symantec root certificate.

Kathleen Wilson

More from Kathleen Wilson »

Categories

- Announcements
- Automated Testing
- BrowserID
- CA Program
- Conferences
- Crypto Engineering
- Firefox
- Firefox OS
- General

https://blog.mozilla.org/security/2018/03/12/distrust-symantec-tls-certificates/

# HTTP attack: TLS strip

- This attack is commonly known as "ssl strip"
- Most servers which support HTTPS implement an HTTP to HTTPS redirect
- When user omits the scheme (i.e., protocol), the browser assumes “http://”
- What if the attacker intercepts the first unencrypted HTTP request?
  - Then they can man-in-the-middle all the traffic to rewrite the HTML to keep the user on the HTTP version of the site

# Redirect HTTP to HTTPS

Client

Server
example.com

```
$ curl -ILs cnn.com | grep "^HTTP\|Location"
HTTP/1.1 301 Moved Permanently
Location: http://www.cnn.com/
HTTP/1.1 301 Moved Permanently
Location: https://www.cnn.com/
HTTP/1.1 200 OK
```

117 Feross Aboukhadijeh


Client
Server
example.com
118 Feross Aboukhadijeh

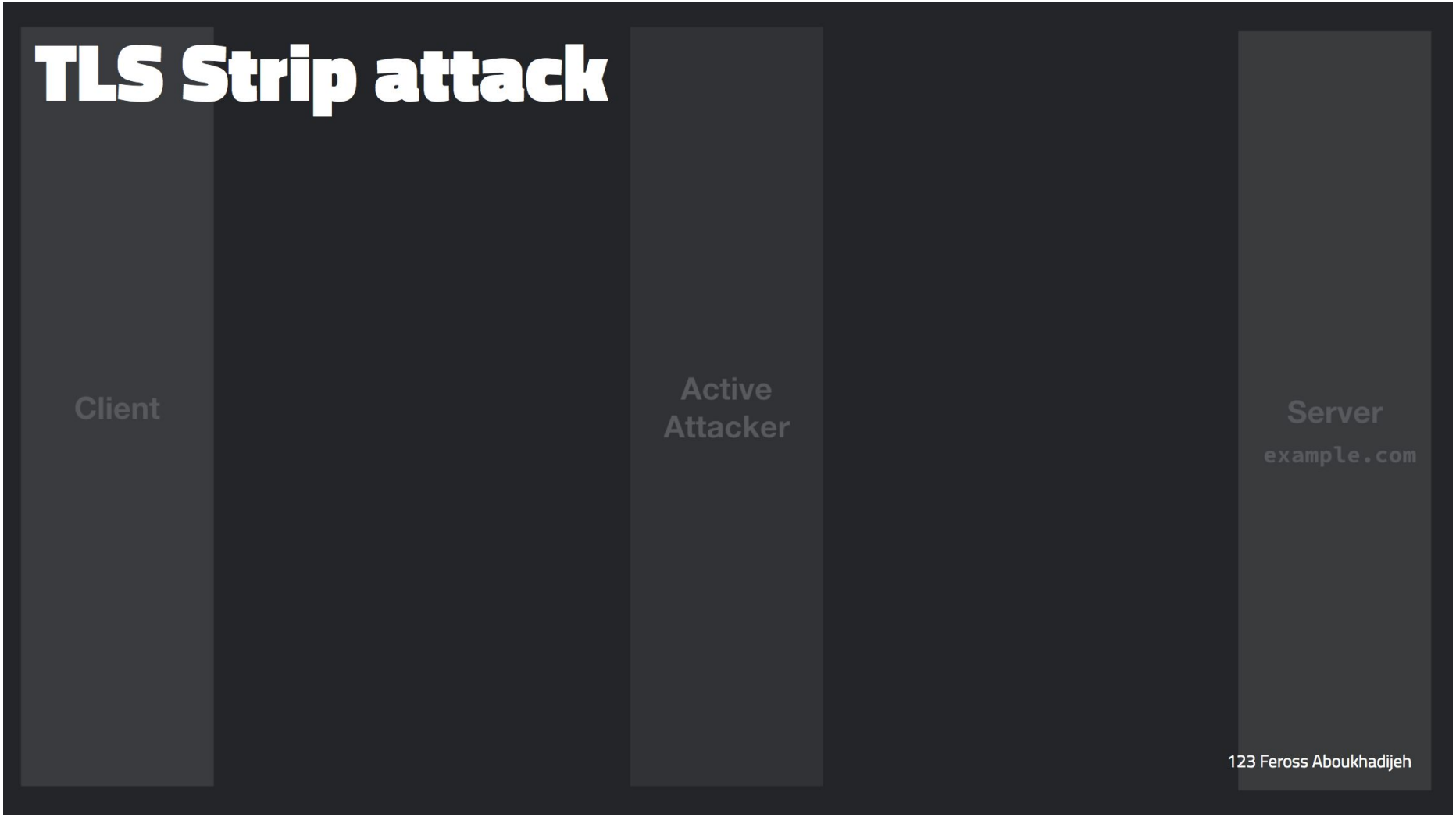

Client
GET / HTTP/1.1
Server
example.com
119 Feross Aboukhadijeh

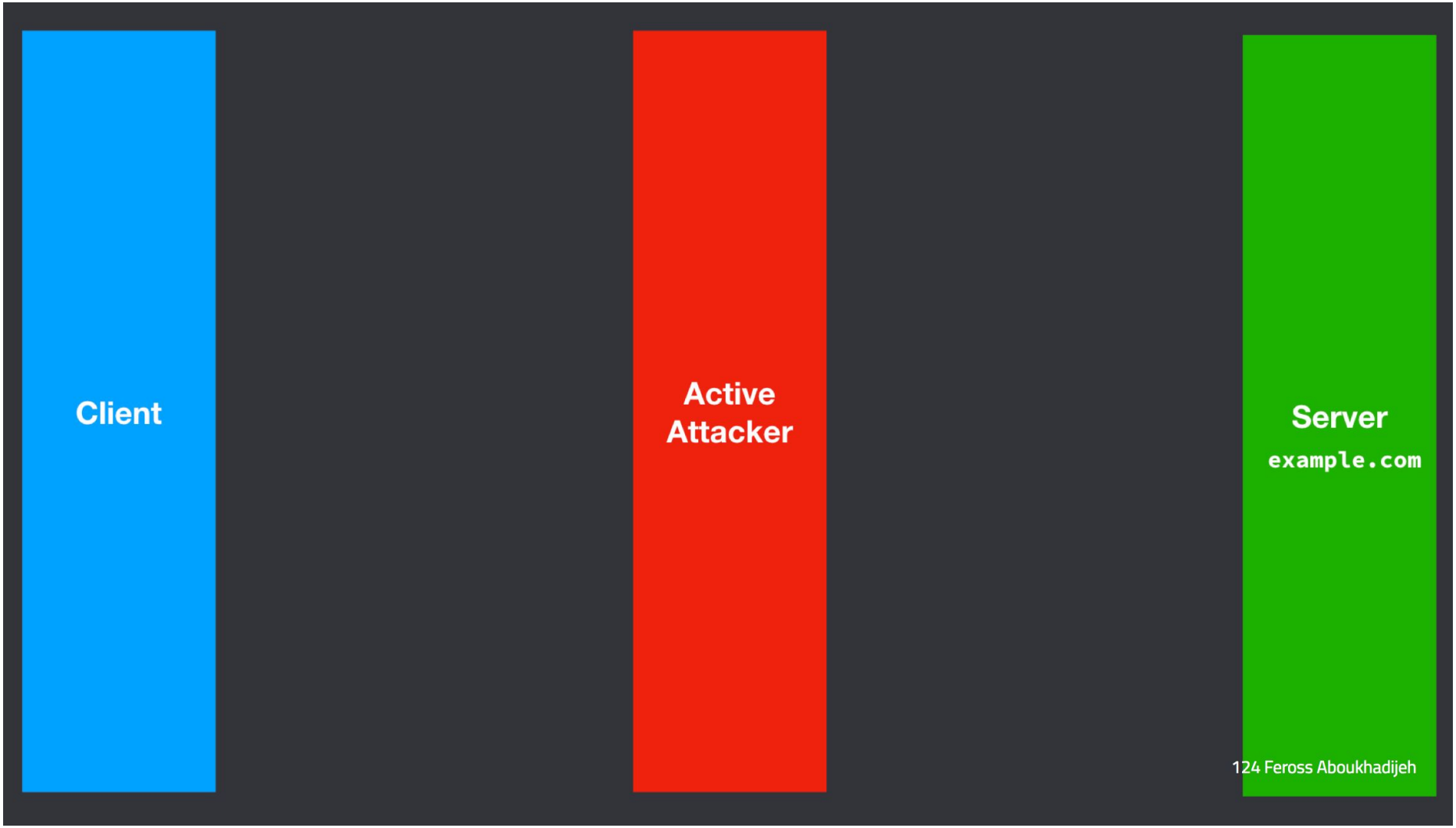

GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
Client
Server
example.com
120 Feross Aboukhadijeh

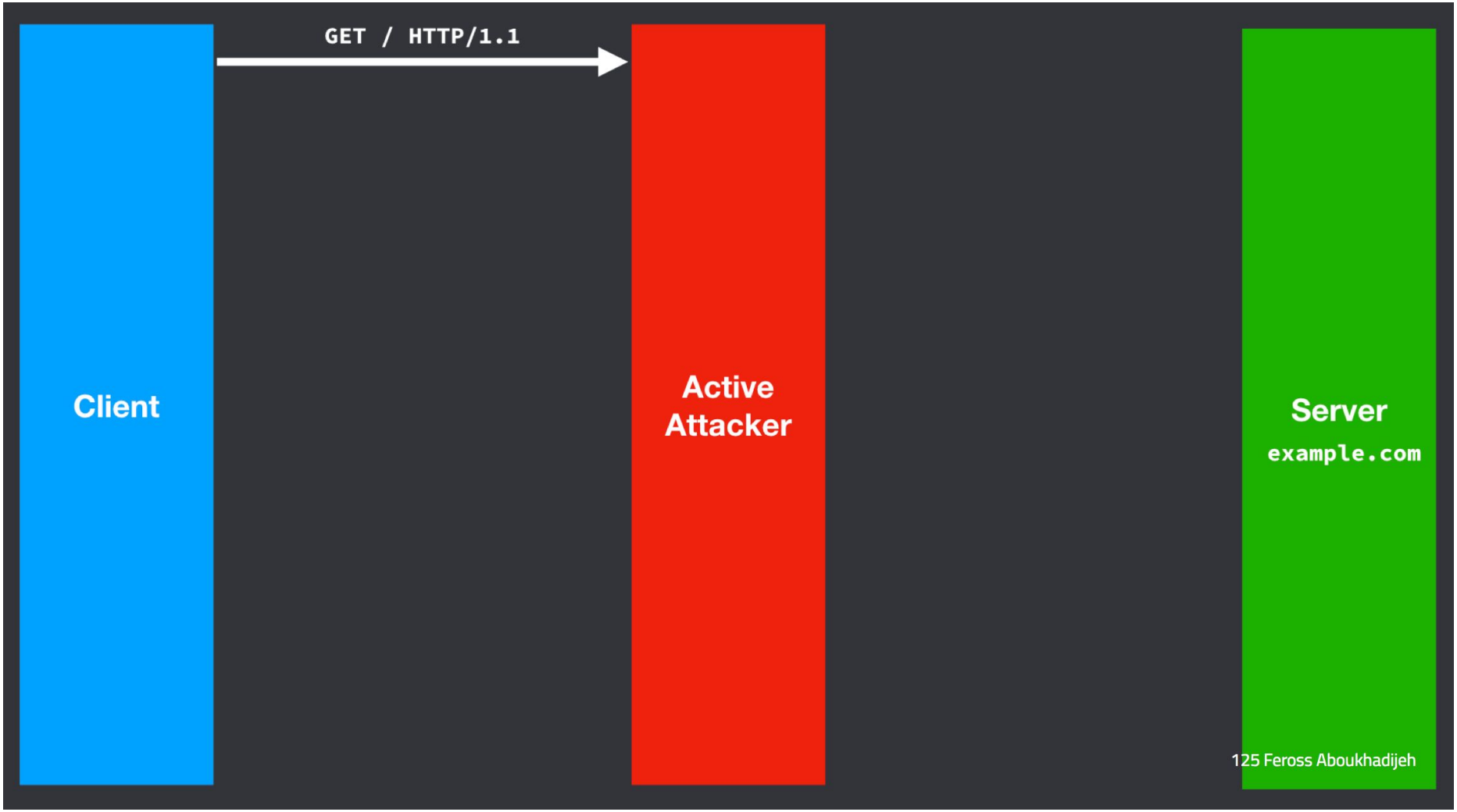

GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
Client
GET / HTTP/1.1
Server
example.com
121 Feross Aboukhadijeh

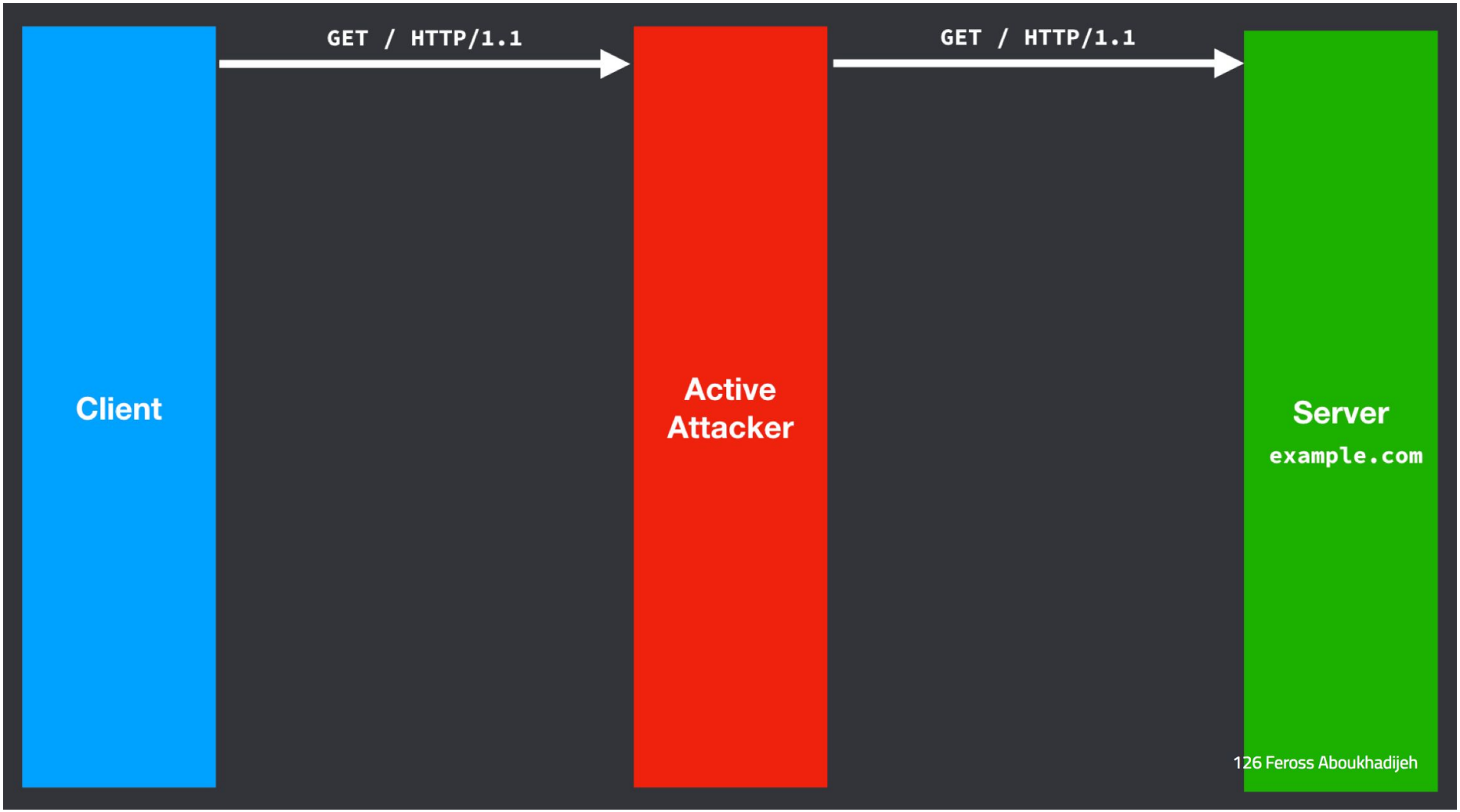

Client
GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> good html
Server
example.com
122 Feross Aboukhadijeh

# TLS Strip attack

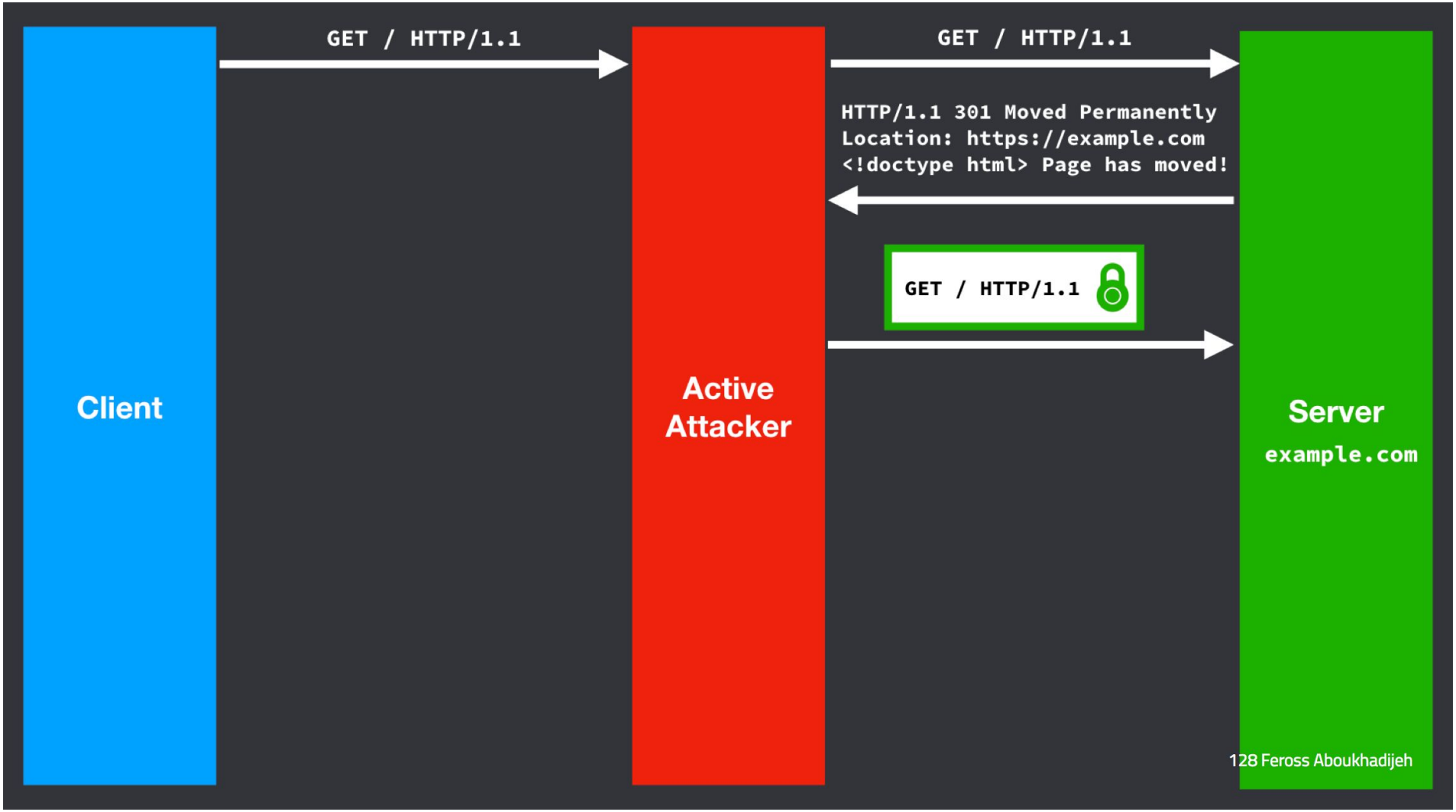


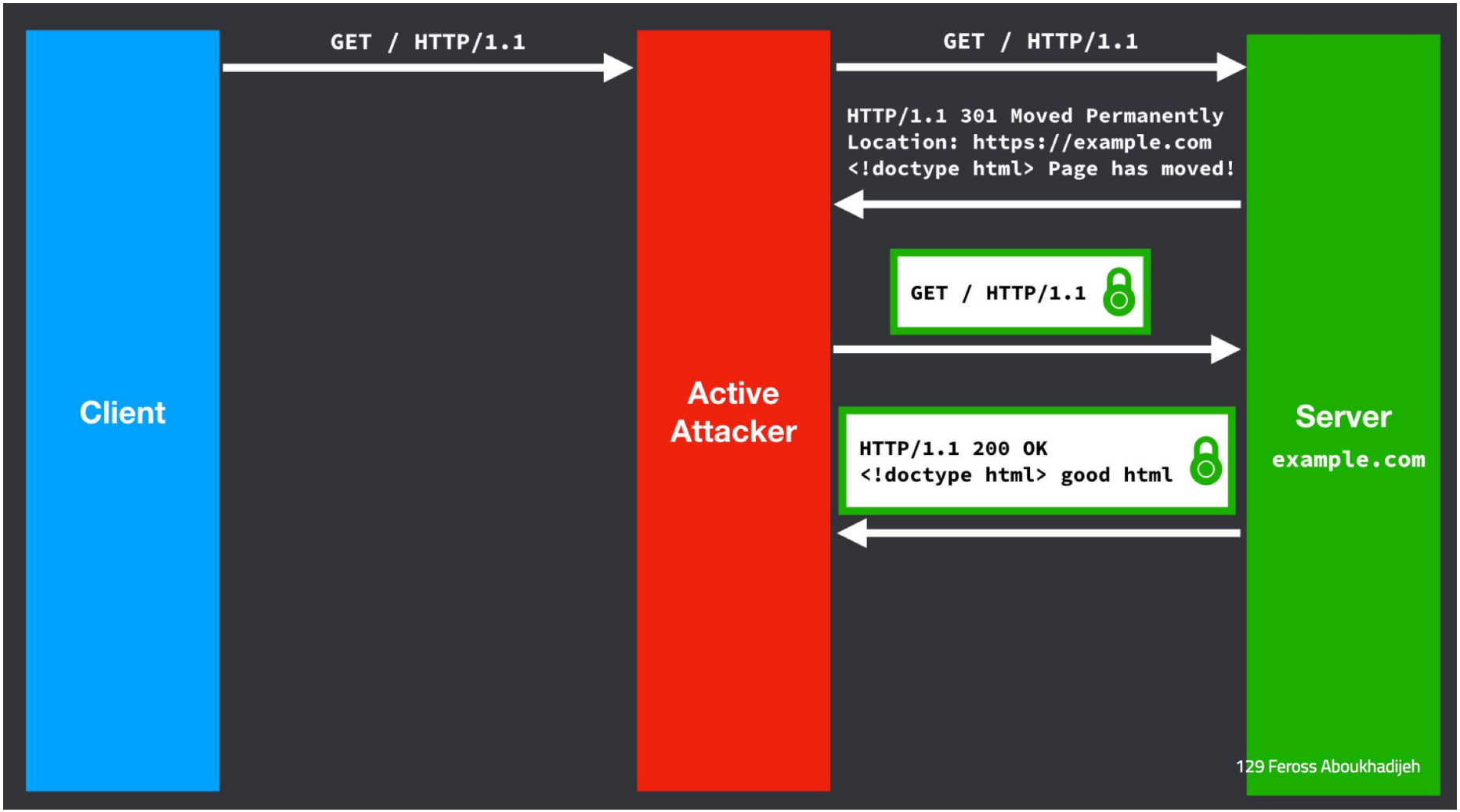

Client
Active Attacker
Server
example.com
124 Feross Aboukhadijeh

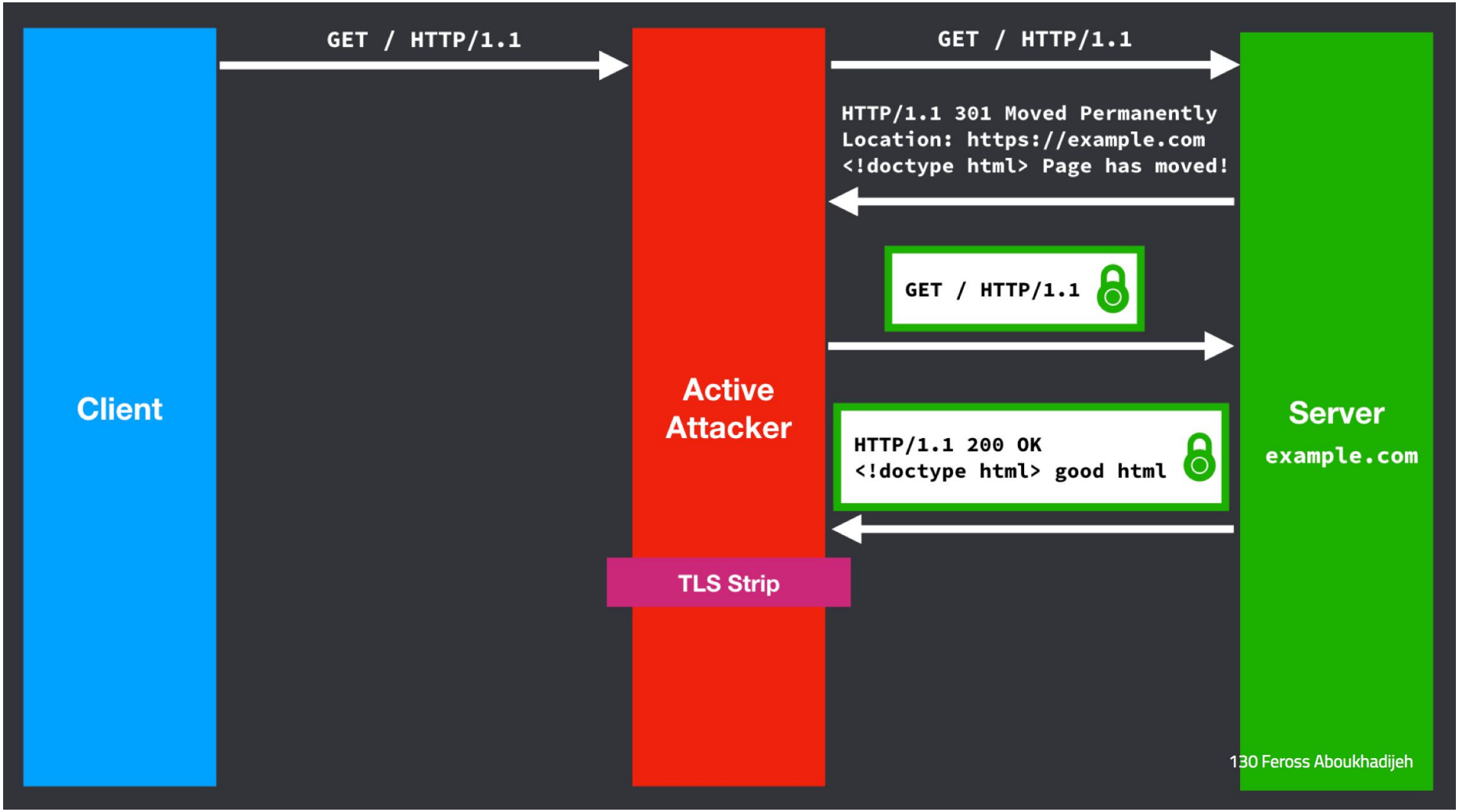

GET / HTTP/1.1
Client
Active Attacker
Server
example.com
125 Feross Aboukhadijeh

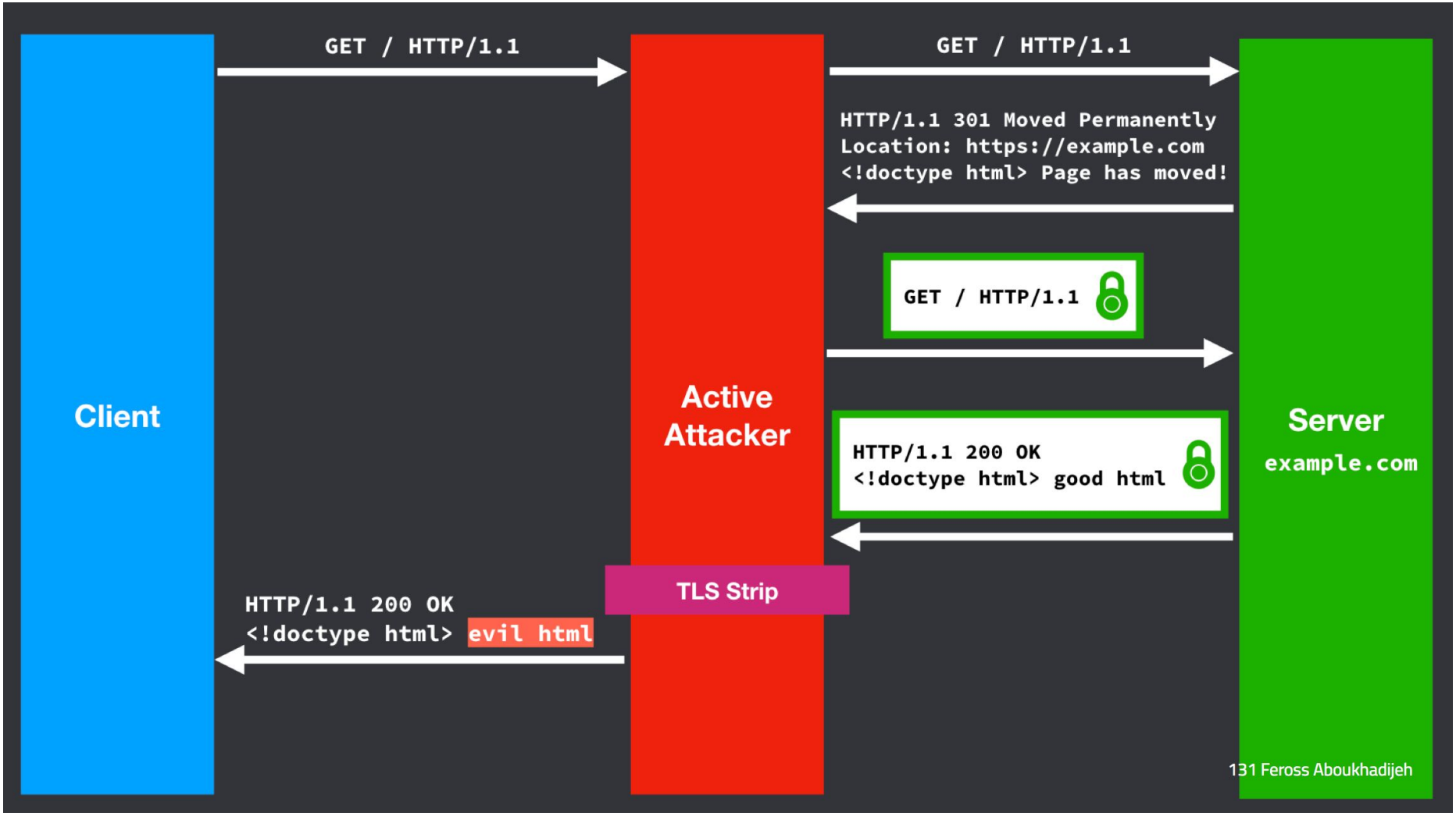

Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
126 Feross Aboukhadijeh

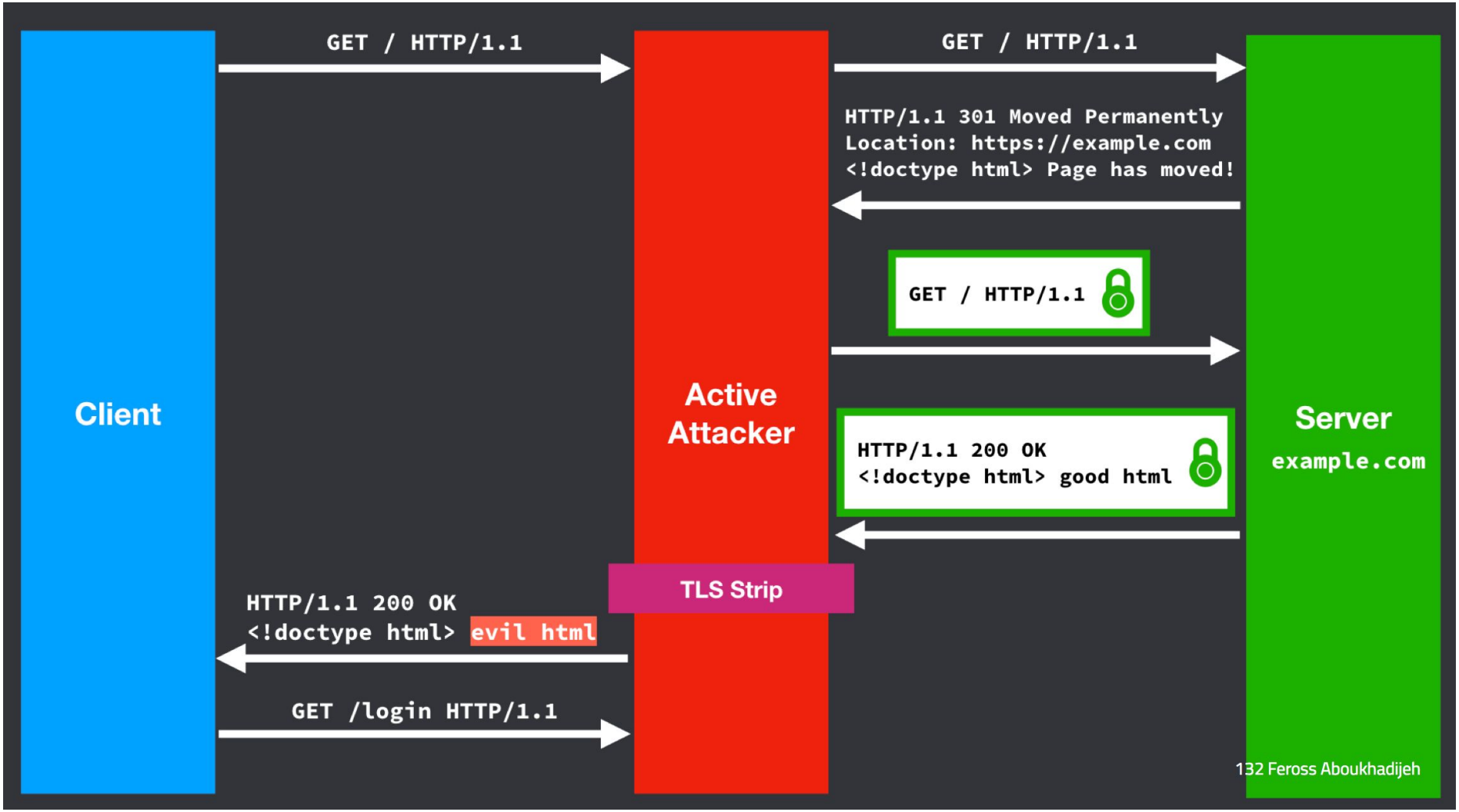

Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
Server
example.com
127 Feross Aboukhadijeh

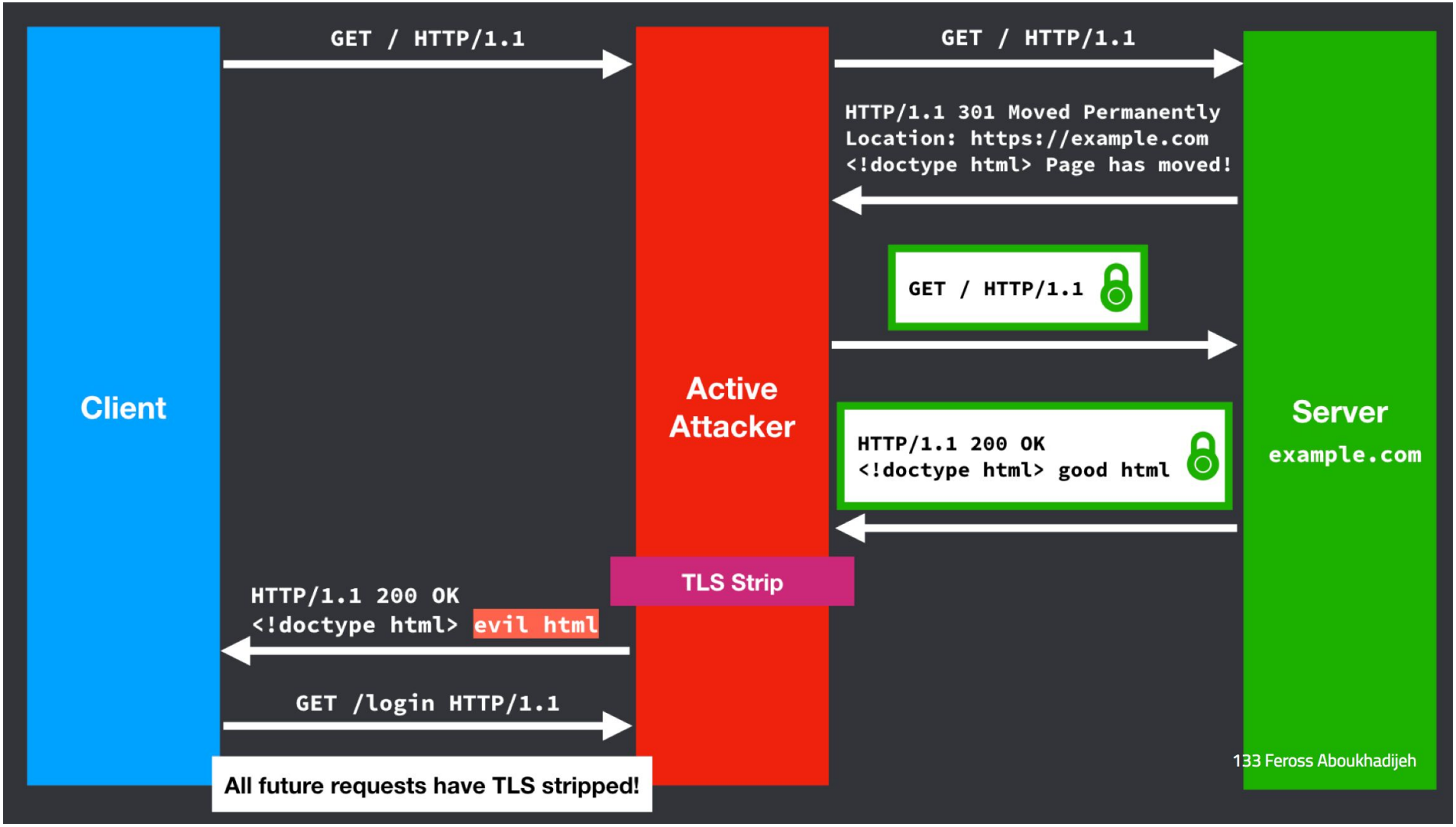

Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
Server
example.com
128 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> good html
Server
example.com
129 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> good html
TLS Strip
130 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> good html
TLS Strip
HTTP/1.1 200 OK
<!doctype html> evil html
131 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> good html
Server
example.com
TLS Strip
HTTP/1.1 200 OK
<!doctype html> evil html
GET /login HTTP/1.1
132 Feross Aboukhadijeh


Client
GET / HTTP/1.1
Active Attacker
GET / HTTP/1.1
Server
example.com
HTTP/1.1 301 Moved Permanently
Location: https://example.com
<!doctype html> Page has moved!
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> good html
TLS Strip
HTTP/1.1 200 OK
<!doctype html> evil html
GET /login HTTP/1.1
All future requests have TLS stripped!
133 Feross Aboukhadijeh

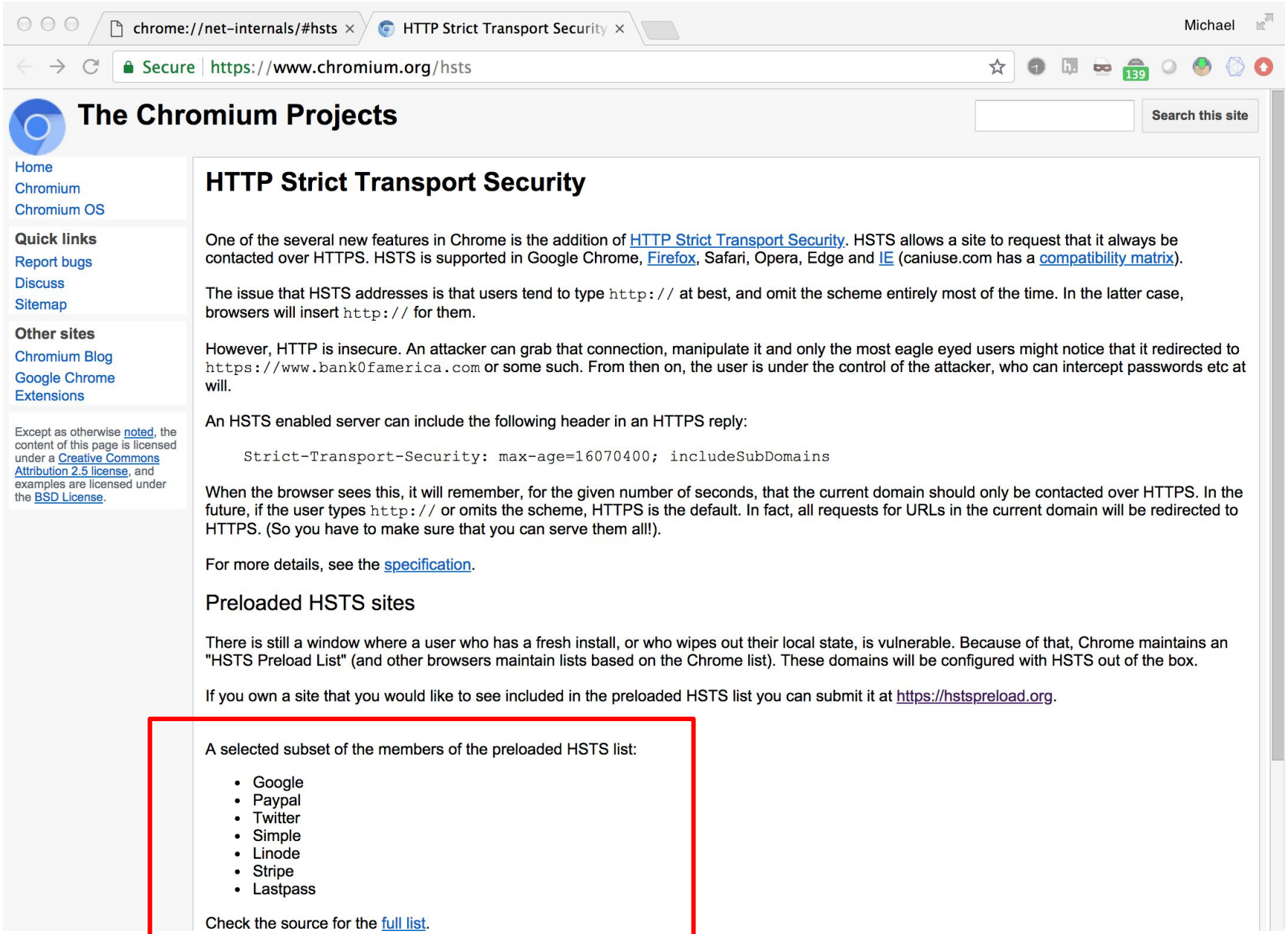
# HTTP strict transport security (HSTS)

- To defend against the TLS Strip attack, the server tells the browser "no matter what protocol the user specifies, always use HTTPS"
- `Strict-Transport-Security: max-age=31536000`
  - Use HTTP header to force browser to use HTTPS for one year!
- Downside: "Trust on first use model" means that first visit to a site is still not secure against man-in-the-middle!
- Should clearing history also clear the HSTS list? Privacy vs. security

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
https://tools.ietf.org/html/rfc6797

# HSTS preload list

- Browsers offer to hardcode sites which want to always be HTTPS only
- `Strict-Transport-Security: max-age=63072000 includeSubDomains; preload`
- Must send includeSubDomains and preload options
- Difficult/impossible to remove a domain once hardcoded into the browser itself
- Certain TLDs added the whole TLD to the preload list (e.g., .dev)


chrome://net-internals/#hsts
HTTP Strict Transport Security
Michael
Secure https://www.chromium.org/hsts
The Chromium Projects
Search this site
Home
Chromium
Chromium OS
Quick links
Report bugs
Discuss
Sitemap
Other sites
Chromium Blog
Google Chrome Extensions
Except as otherwise noted, the content of this page is licensed under a Creative Commons Attribution 2.5 license, and examples are licensed under the BSD License.
HTTP Strict Transport Security
One of the several new features in Chrome is the addition of HTTP Strict Transport Security. HSTS allows a site to request that it always be contacted over HTTPS. HSTS is supported in Google Chrome, Firefox, Safari, Opera, Edge and IE (caniuse.com has a compatibility matrix).
The issue that HSTS addresses is that users tend to type http:// at best, and omit the scheme entirely most of the time. In the latter case, browsers will insert http:// for them.
However, HTTP is insecure. An attacker can grab that connection, manipulate it and only the most eagle eyed users might notice that it redirected to https://www.bank0famerica.com or some such. From then on, the user is under the control of the attacker, who can intercept passwords etc at will.
An HSTS enabled server can include the following header in an HTTPS reply:
Strict-Transport-Security: max-age=16070400; includeSubDomains
When the browser sees this, it will remember, for the given number of seconds, that the current domain should only be contacted over HTTPS. In the future, if the user types http:// or omits the scheme, HTTPS is the default. In fact, all requests for URLs in the current domain will be redirected to HTTPS. (So you have to make sure that you can serve them all!).
For more details, see the specification.
Preloaded HSTS sites
There is still a window where a user who has a fresh install, or who wipes out their local state, is vulnerable. Because of that, Chrome maintains an "HSTS Preload List" (and other browsers maintain lists based on the Chrome list). These domains will be configured with HSTS out of the box.
If you own a site that you would like to see included in the preloaded HSTS list you can submit it at https://hstspreload.org.
A selected subset of the members of the preloaded HSTS list:
Google
Paypal
Twitter
Simple
Linode
Stripe
Lastpass
Check the source for the full list.

# Uncovered issues

- Public Key Pinning (HPKP)
- Certificate Transparency
- DNS Certification Authority Authorization (DNS CAA)