6.1800 Spring 2024

Lecture #24: Tor

what to do when secure channels aren't enough

Katrina LaCurts | lacurts@mit.edu | 6.1800 2024

6.1800 in the news

Maximum-severity GitLab flaw allowing account hijacking under active exploitation

The threat is potentially grave because it could be used in supply-chain attacks.

DAN GOODIN - 5/2/2024, 3:02 PM

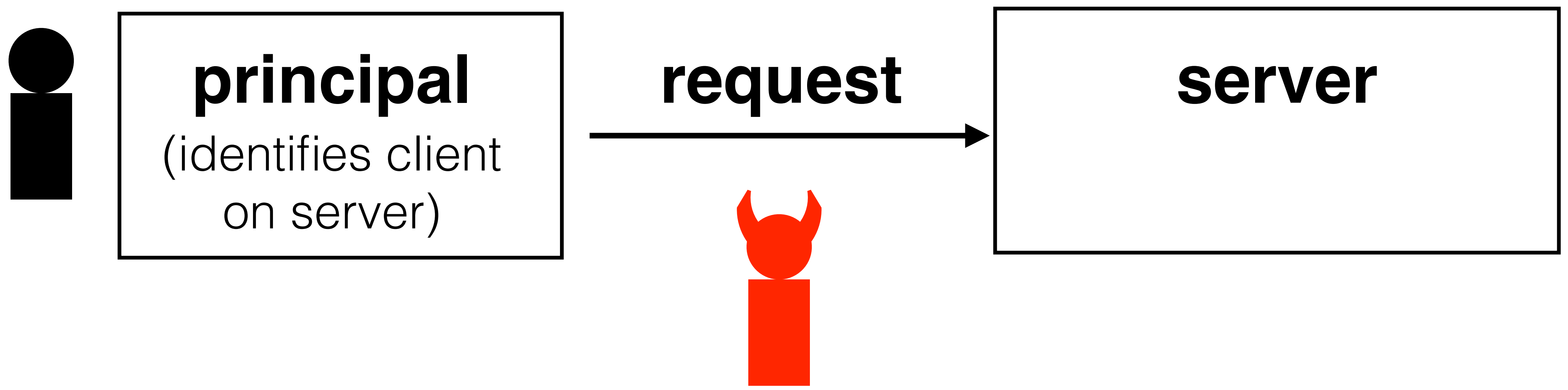
A maximum severity vulnerability that allows hackers to hijack GitLab accounts with no user interaction required is now under active exploitation, federal government officials warned as data showed that thousands of users had yet to install a patch released in January.

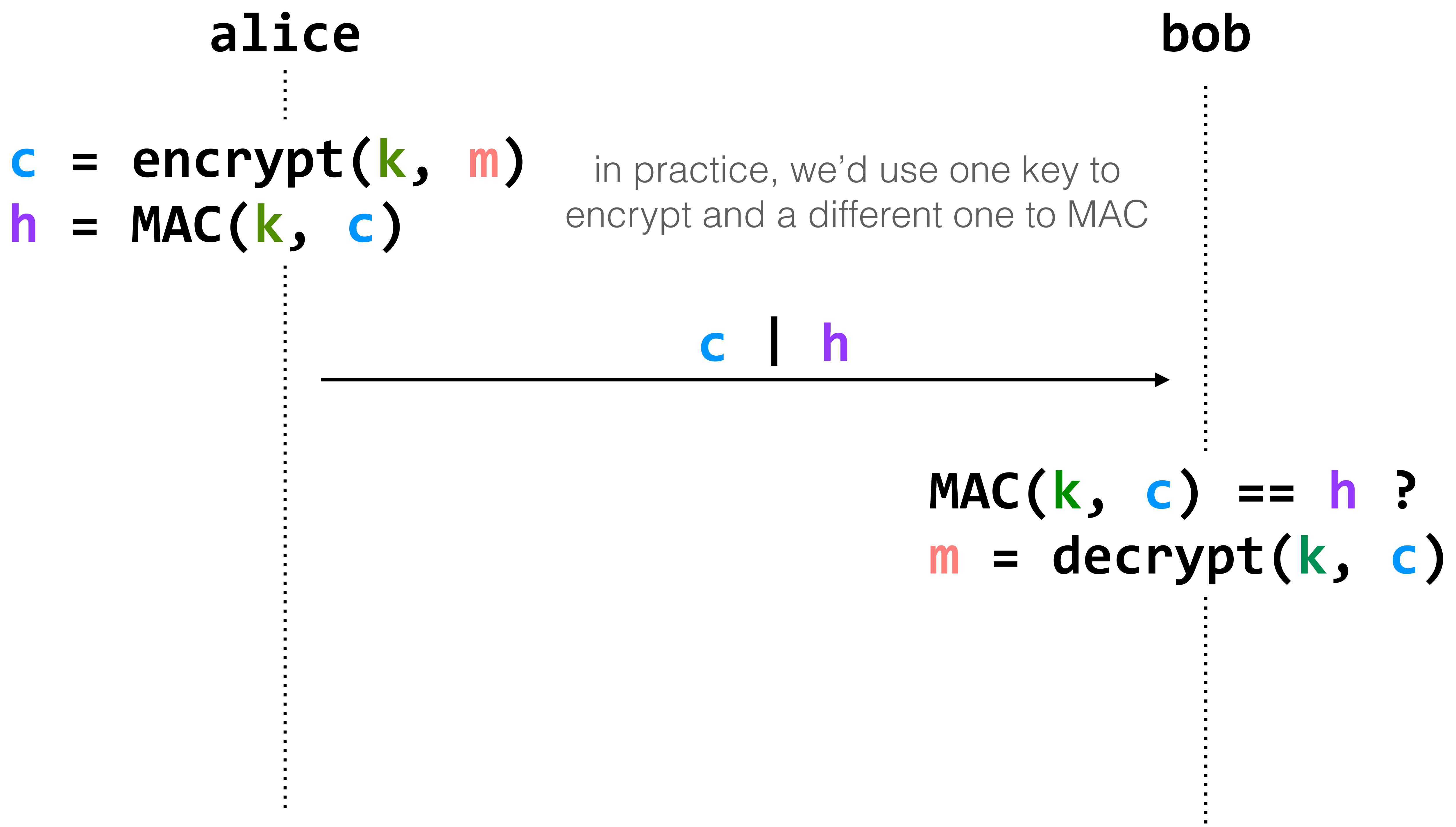
A change GitLab implemented in May 2023 made it possible for users to initiate password changes through links sent to secondary email addresses. The move was designed to permit resets when users didn't have access to the email address used to establish the account. In January, GitLab disclosed that the feature allowed attackers to send reset emails to accounts they controlled and from there click on the embedded link and take over the account.

note that the attack does **not** exploit how passwords are stored, but instead how the system allows them to be reset

https://arstechnica.com/security/2024/05/0-click-gitlab-hijacking-flaw-under-active-exploit-with-thousands-still-unpatched/

today, we're still considering adversaries that are observing data on the network





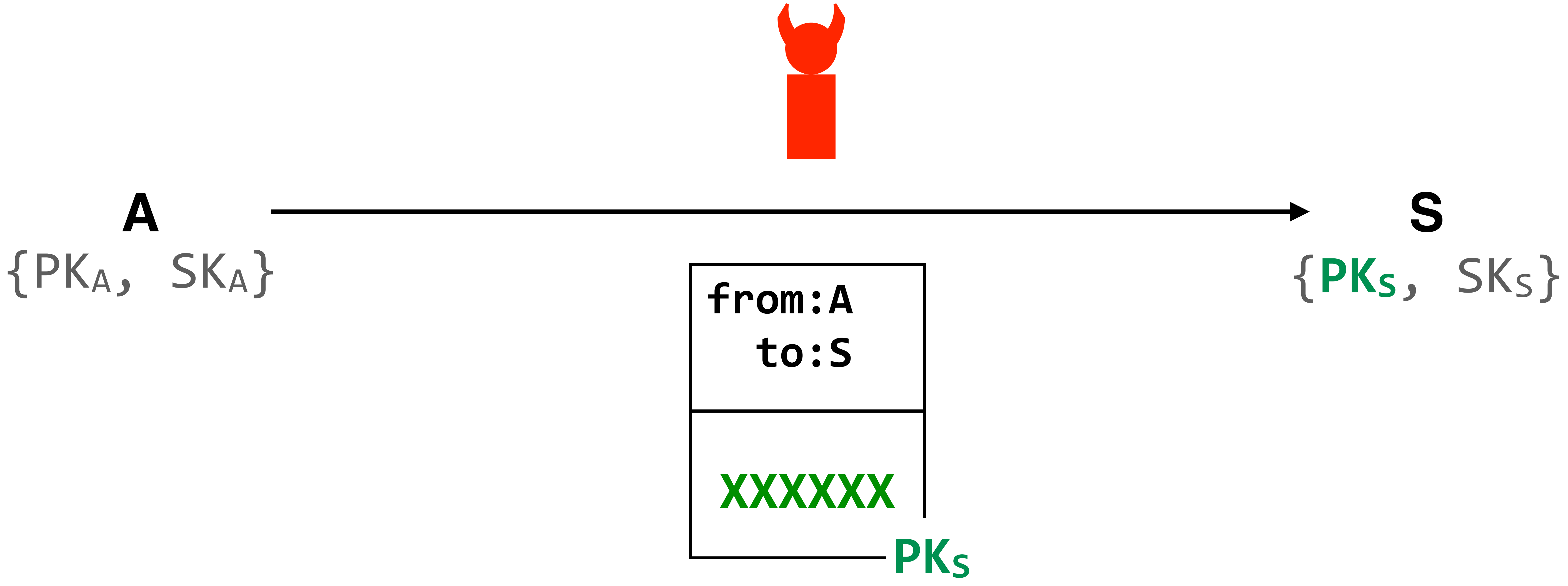
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```

this is different from how you saw public-key cryptography used for signatures, and different from how you saw symmetric keys used for encryption



alice is encrypting data **to** S using its public key

problem: packet header exposes to the adversary that **A** is communicating with **S**

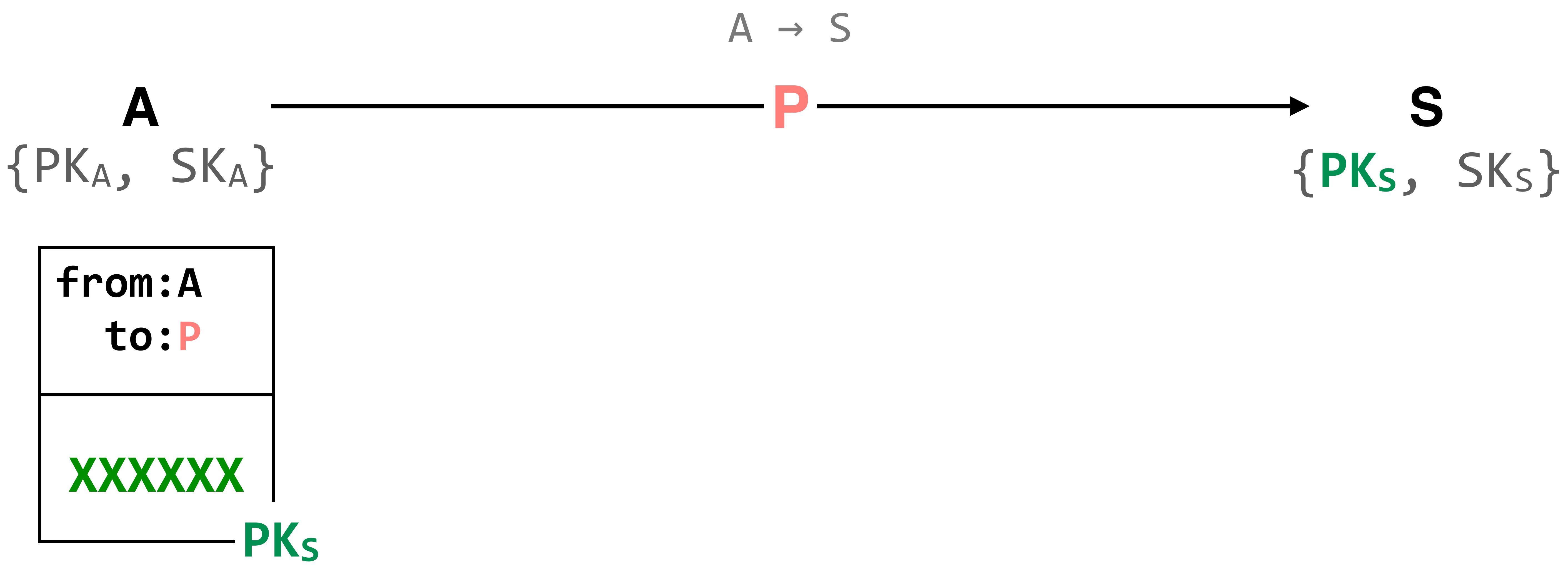
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$encrypt(PK_x, m) = c$

$decrypt(SK_x, c) = m$



things to avoid

no packet should say "from: A; to: S"

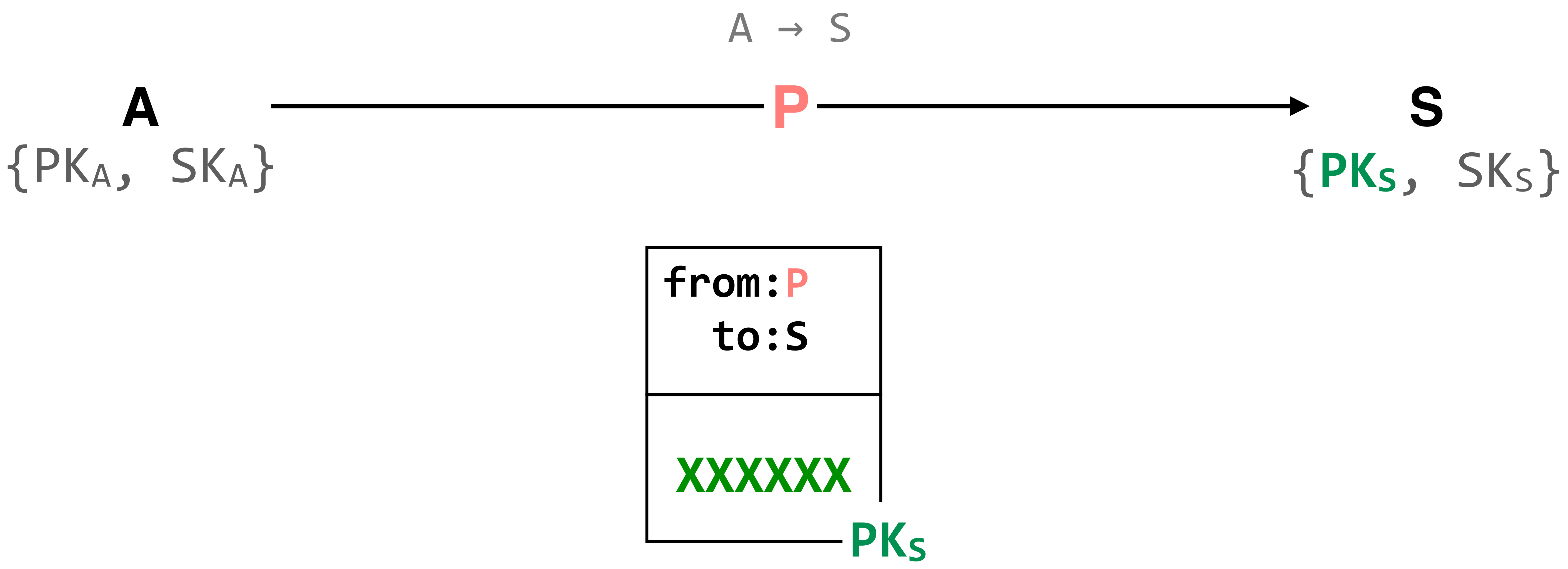
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$encrypt(PK_x, m) = c$$

$$decrypt(SK_x, c) = m$$



things to avoid

no packet should say "from: A; to: S"

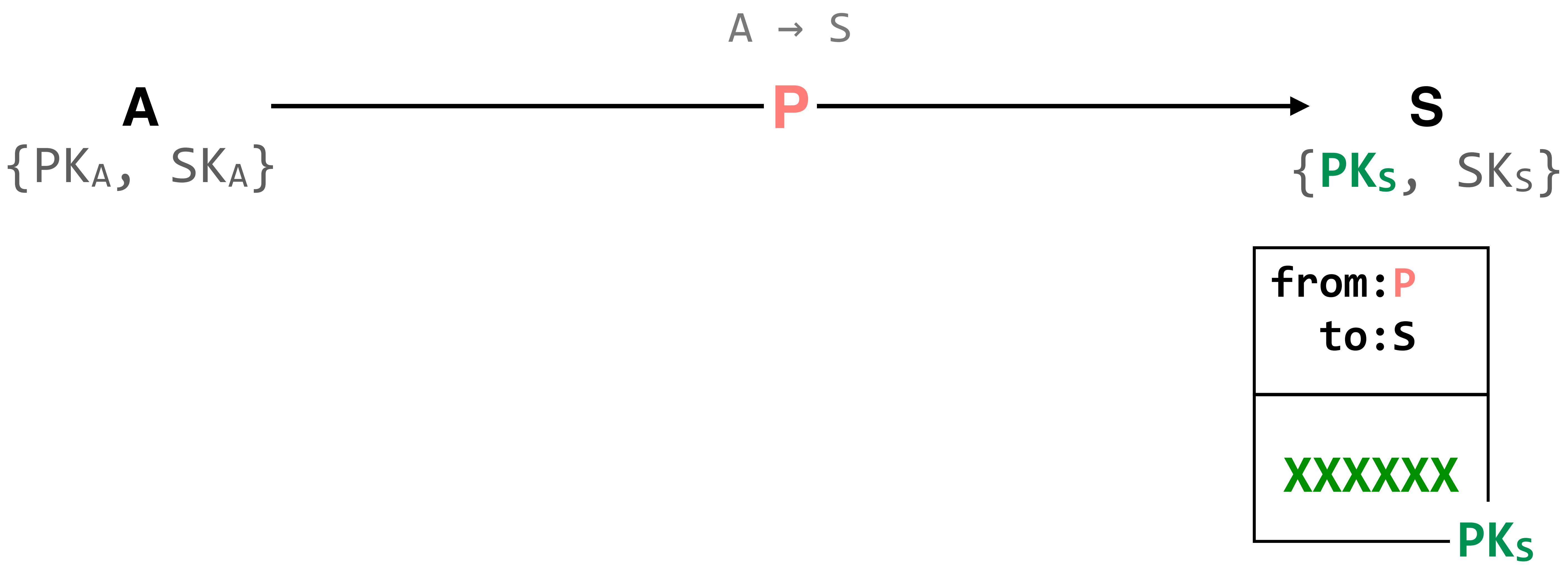
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$\texttt{encrypt}(PK_x, m) = c$$

$$\texttt{decrypt}(SK_x, c) = m$$



things to avoid

- no packet should say "from: A; to: S"
- no entity in the network should receive a packet from A and send it directly to S
- no entity in the network should keep *state* that links A to S

problem: P knows that **A** is communicating with **S**

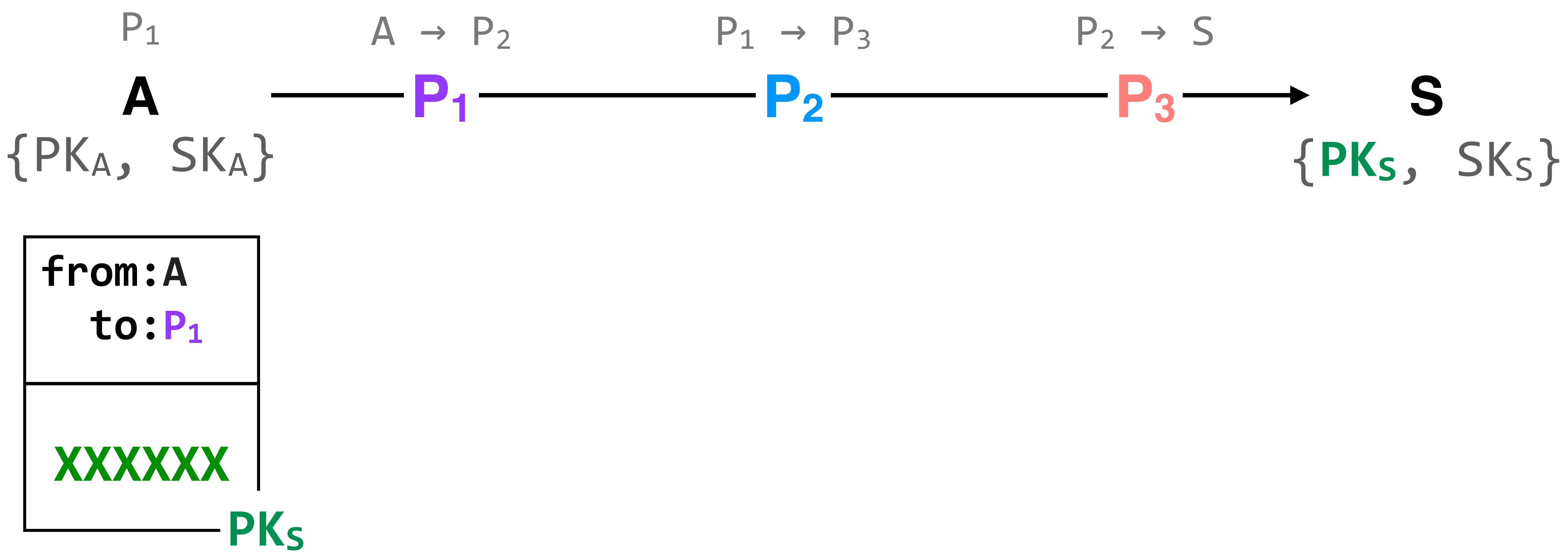
a single proxy alone *can* be useful for other things; we'll return to this later in the lecture.

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

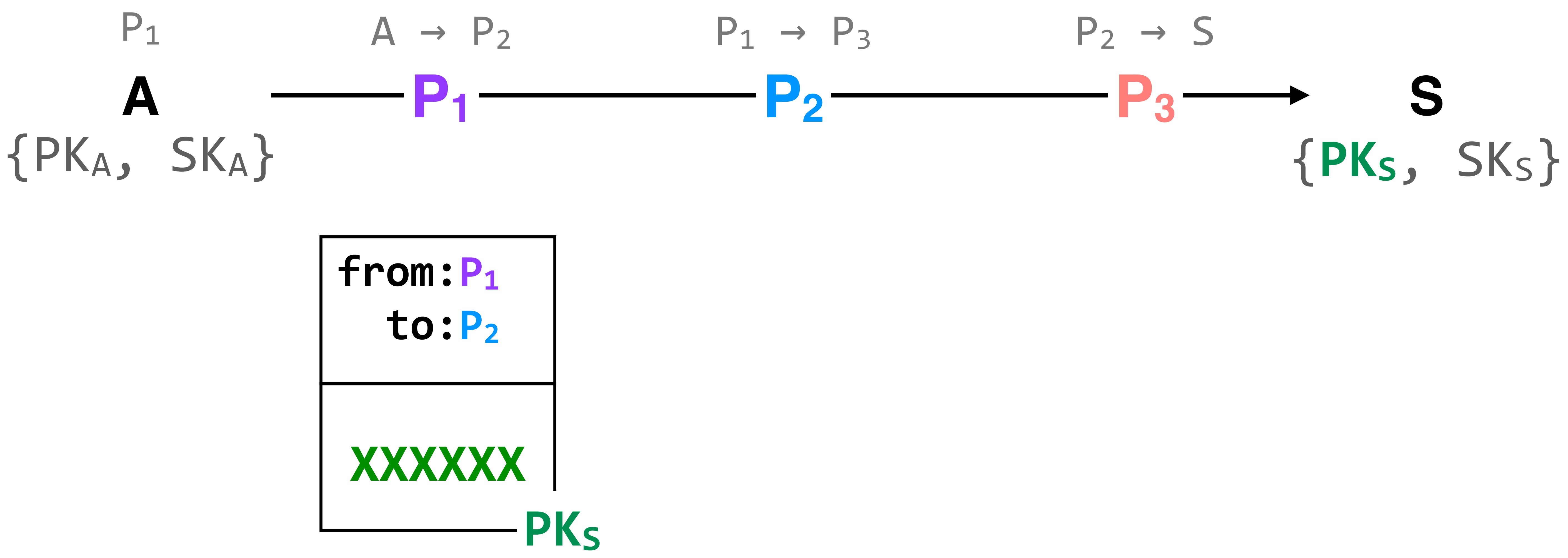
no entity in the network should keep *state* that links A to S

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

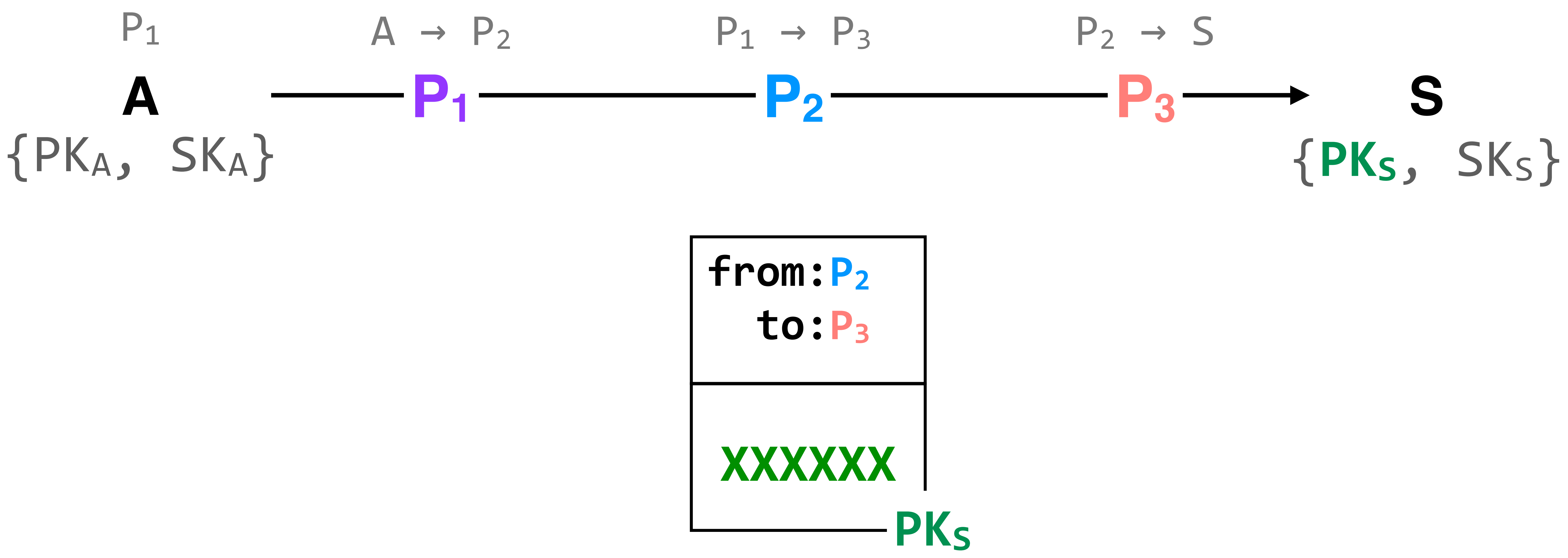
no entity in the network should keep *state* that links A to S

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

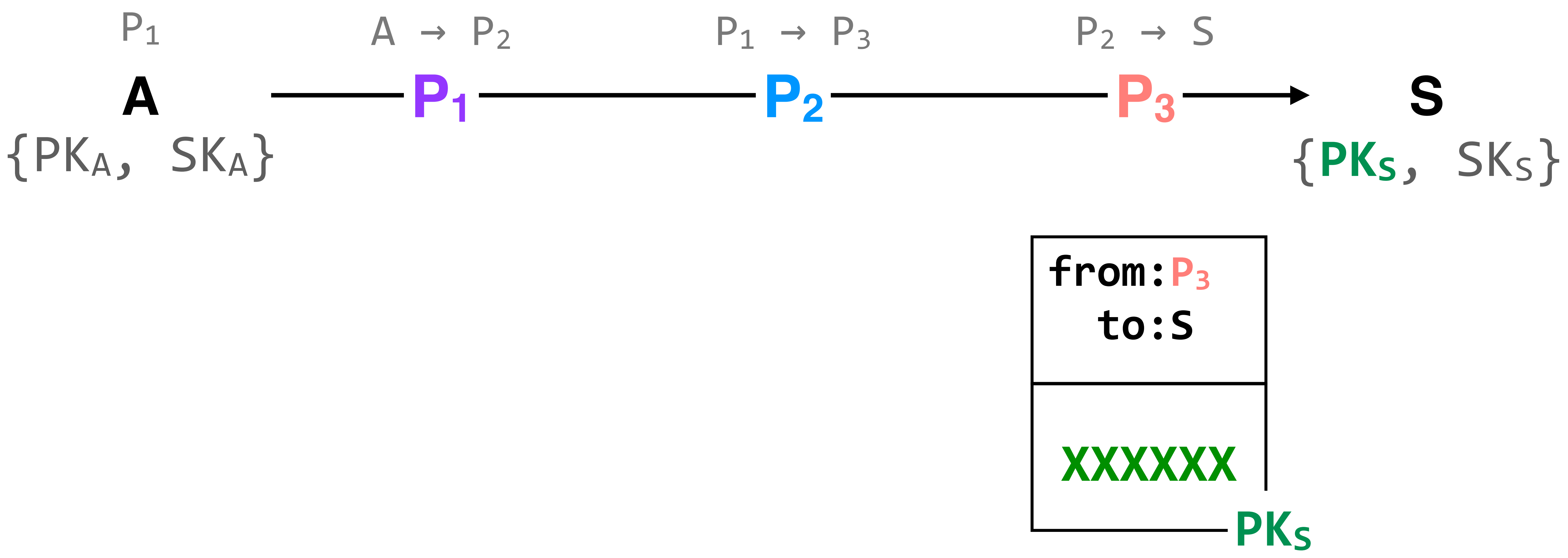
no entity in the network should keep *state* that links A to S

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

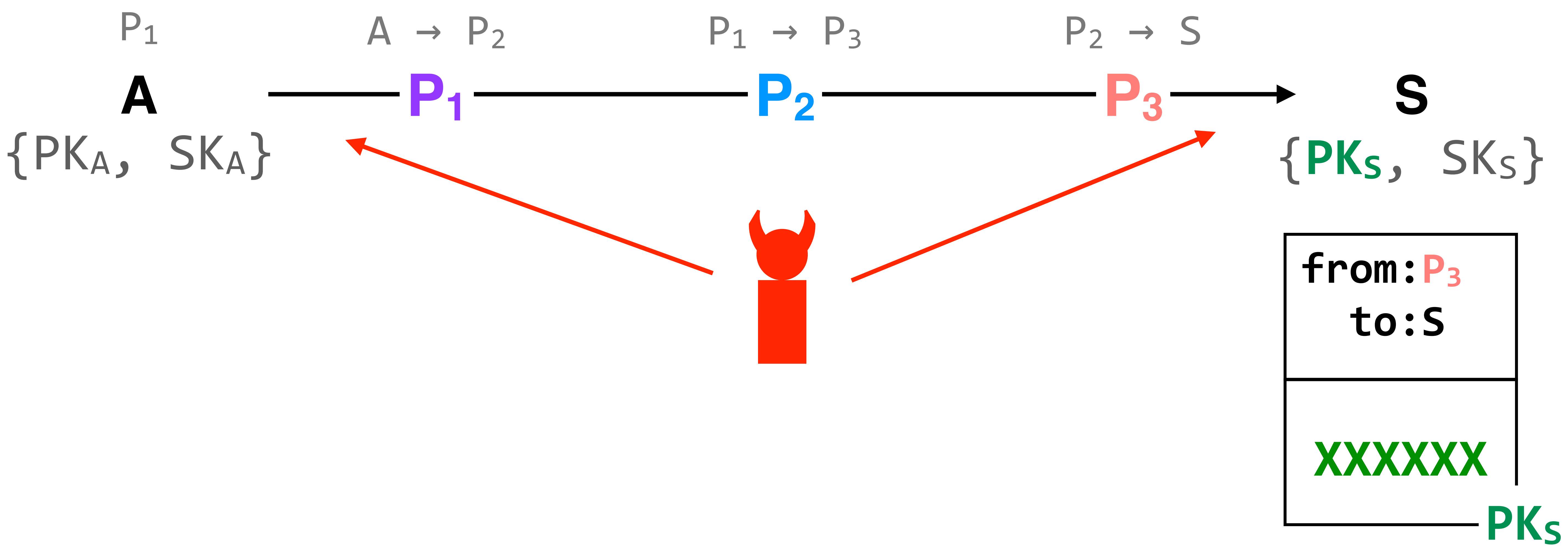
no entity in the network should keep *state* that links A to S

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



things to avoid

- 👍 no packet should say "from: A; to: S"
- 👍 no entity in the network should receive a packet from A and send it directly to S
- 👍 no entity in the network should keep *state* that links A to S
- data should not appear the same across packets

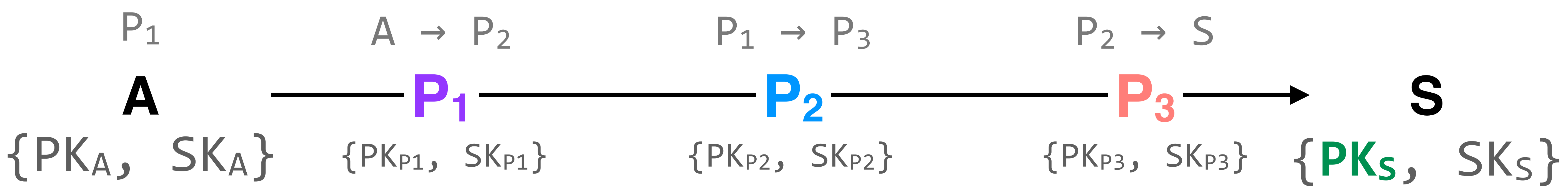
problem: an adversary with multiple vantage points can observe the same data traveling from **A** to **S**

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

no entity in the network should keep *state* that links A to S

data should not appear the same across packets

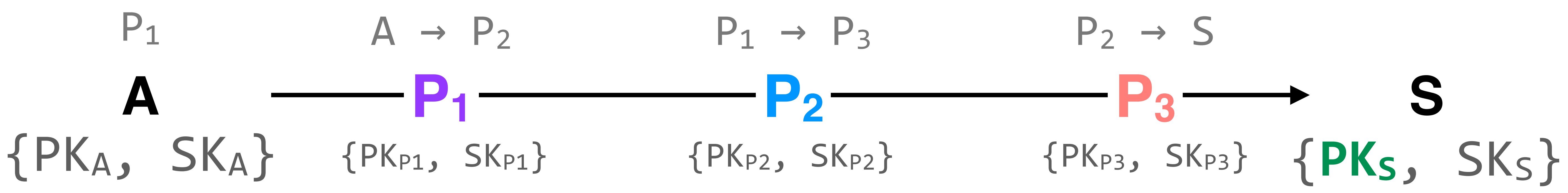
onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



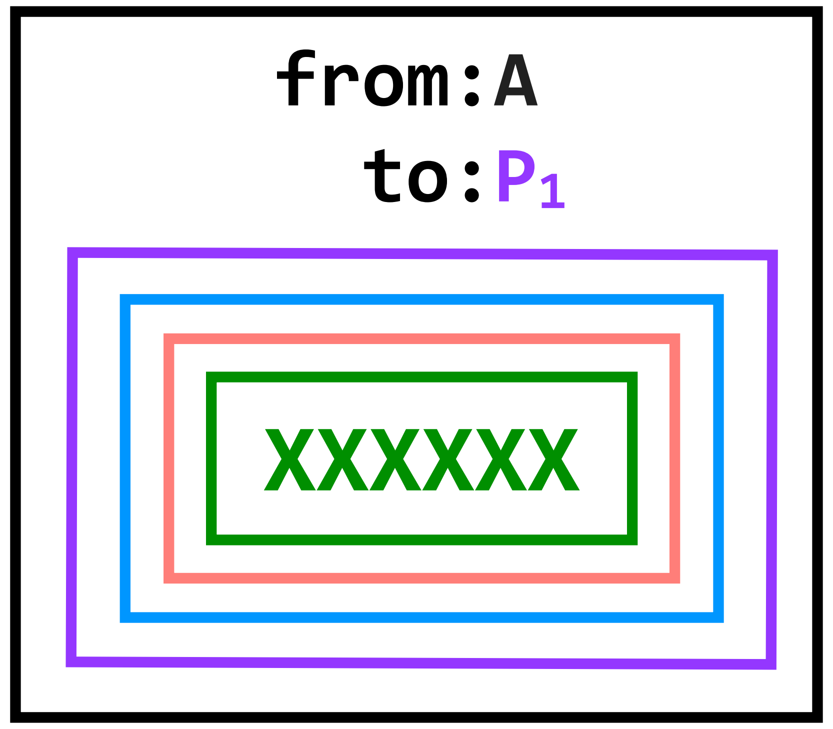
things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

no entity in the network should keep *state* that links A to S

data should not appear the same across packets



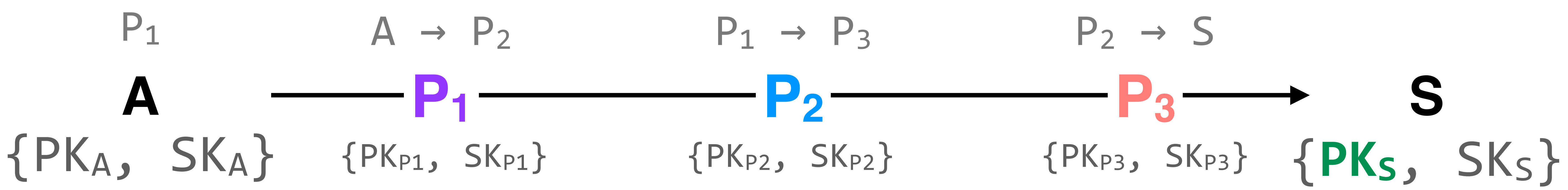
onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



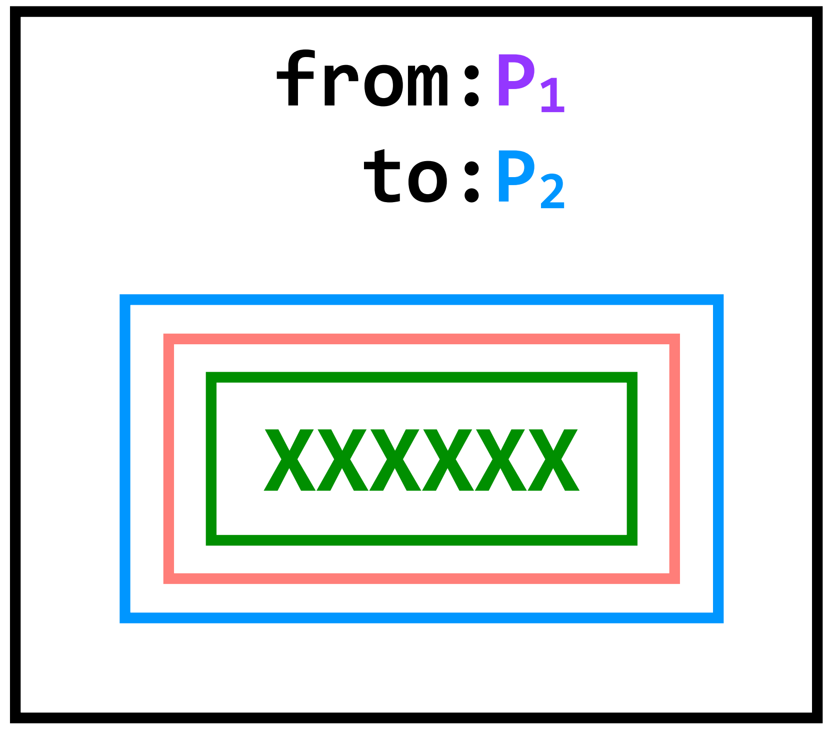
things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

no entity in the network should keep *state* that links A to S

data should not appear the same across packets



onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

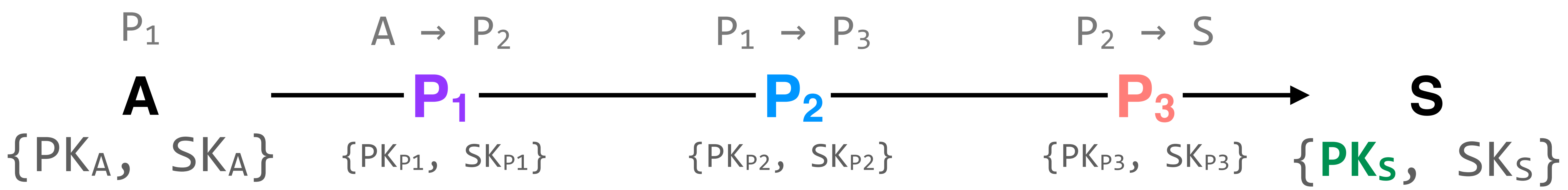
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$\texttt{encrypt}(PK_x, m) = c$$

$$\texttt{decrypt}(SK_x, c) = m$$



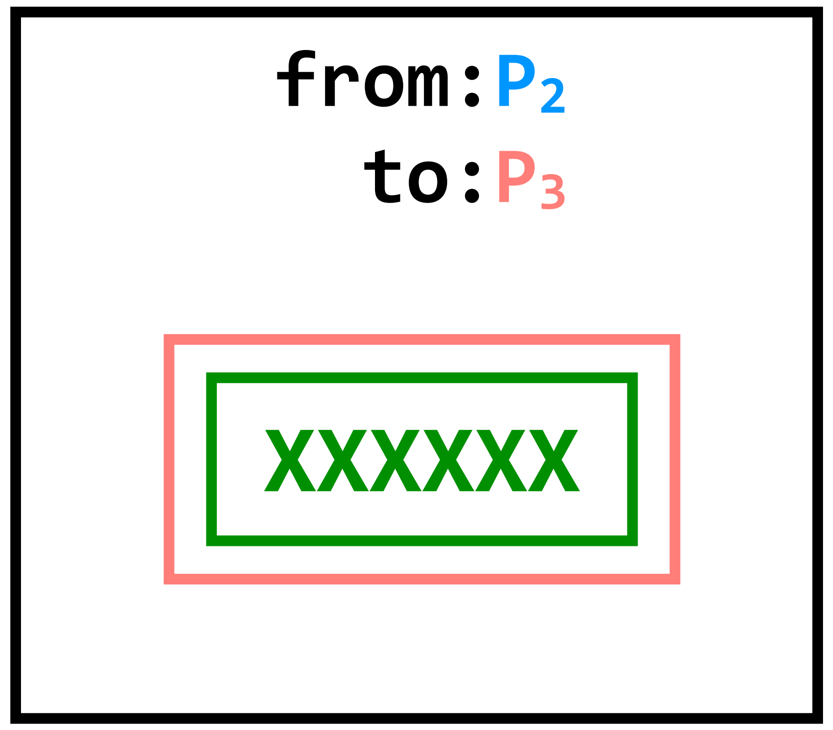
things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

no entity in the network should keep *state* that links A to S

data should not appear the same across packets



onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

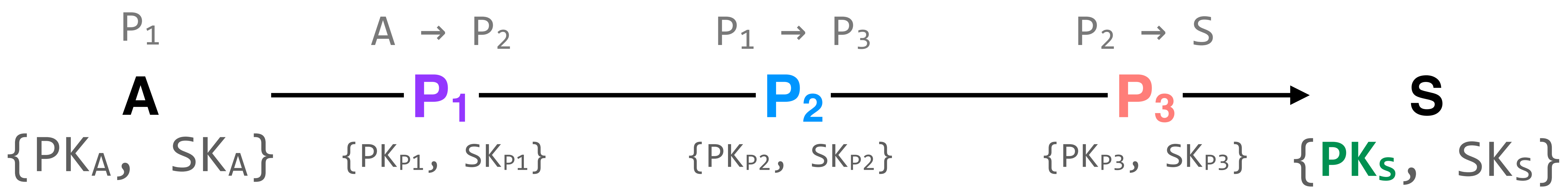
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

encrypt(PK_x, m) = c

decrypt(SK_x, c) = m



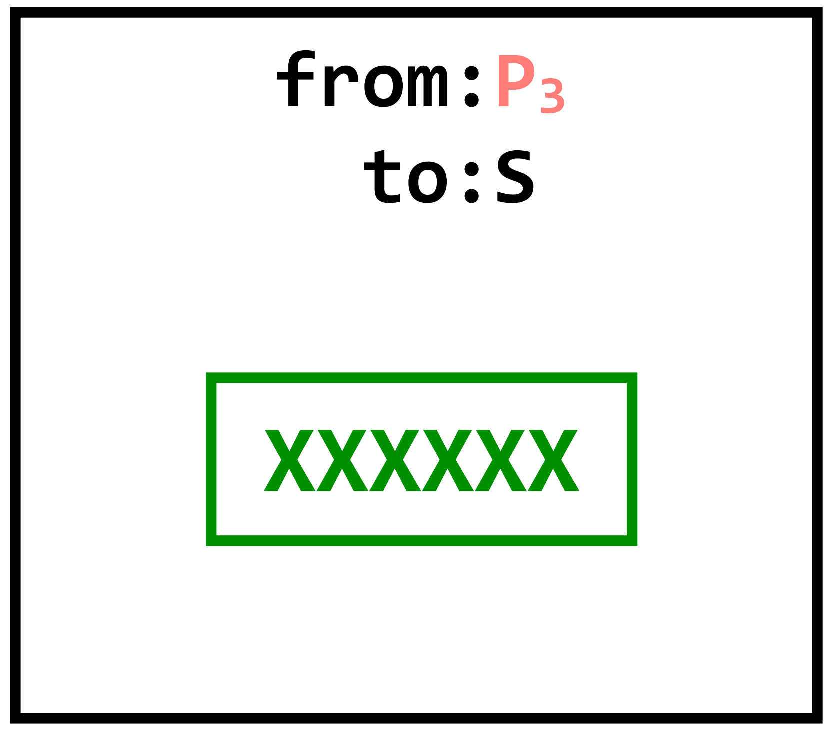
things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

no entity in the network should keep *state* that links A to S

data should not appear the same across packets



onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

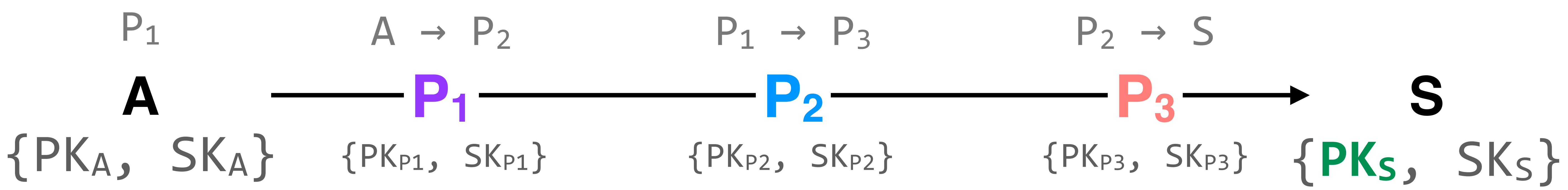
policy: provide **anonymity** (only the client should know that they're communicating with the server)

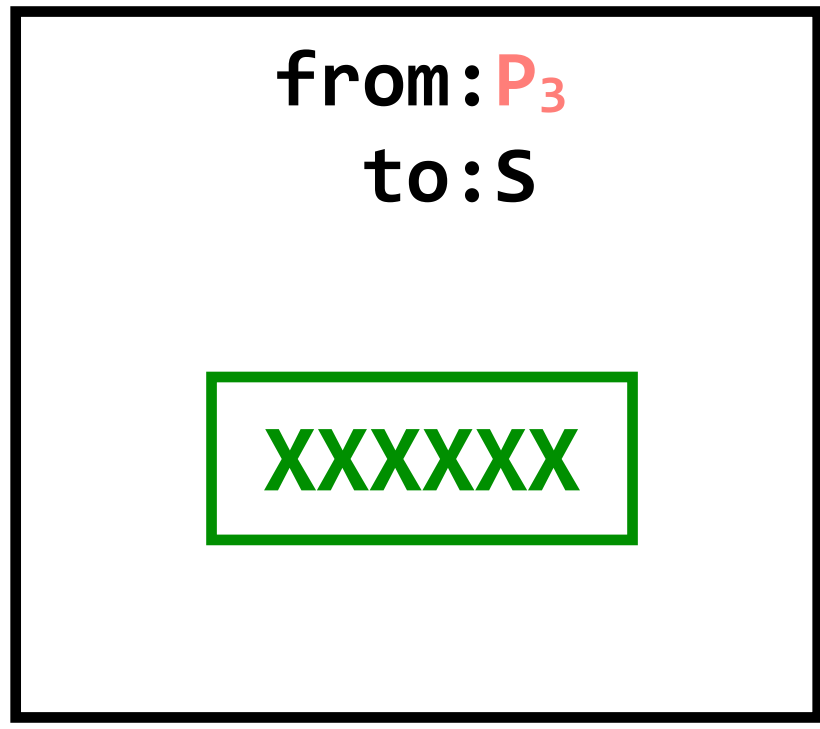
threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$\texttt{encrypt}(PK_x, m) = c$$

$$\texttt{decrypt}(SK_x, c) = m$$





things to avoid

no packet should say "from: A; to: S"

no entity in the network should receive a packet from A and send it directly to S

no entity in the network should keep *state* that links A to S

data should not appear the same across packets

onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

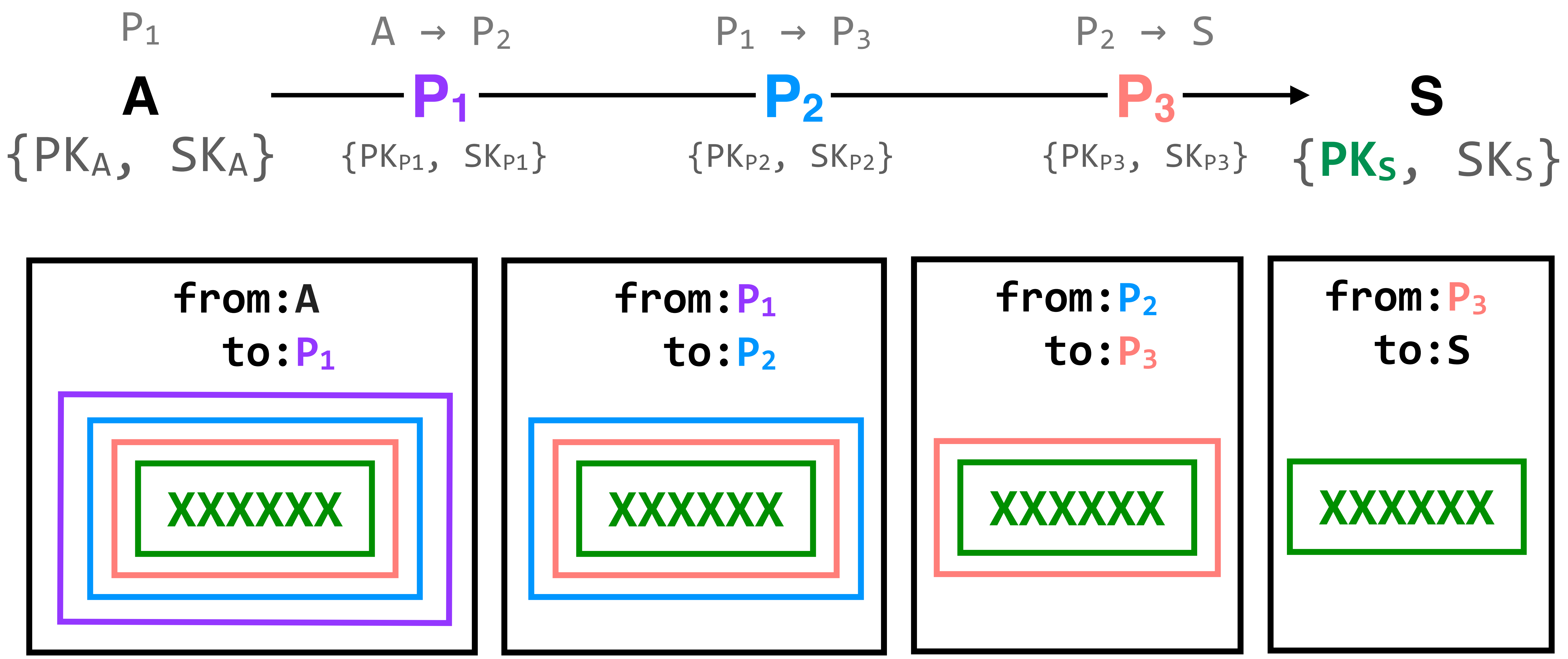
public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$encrypt(PK_x, m) = c$$

$$decrypt(SK_x, c) = m$$

things to avoid

- 👍 no packet should say "from: A; to: S"
- 👍 no entity in the network should receive a packet from A and send it directly to S
- 👍 no entity in the network should keep *state* that links A to S
- 👍 data should not appear the same across packets



onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

in practice, **tor** uses public-key cryptography to securely exchange **symmetric keys** between A and each node in the circuit, and the layers of encryption use those symmetric keys; this is what allow traffic to travel in both directions

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

Am I totally anonymous if I use Tor?

Generally it is impossible to have perfect anonymity, even with Tor. Though there are some things you can practice to improve your anonymity while using Tor and offline.

Use Tor Browser and software specifically configured for Tor

Tor does not protect all of your computer's Internet traffic when you run it. Tor only protects applications that are properly configured to send their Internet traffic through Tor.

Web browsing:

- Safe: Tor Browser
- Unsafe: Any other browser configured to use Tor as a proxy

File sharing:

- Safe: OnionShare
- Unsafe: BitTorrent over Tor

https://support.torproject.org/faq/staying-anonymous/

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

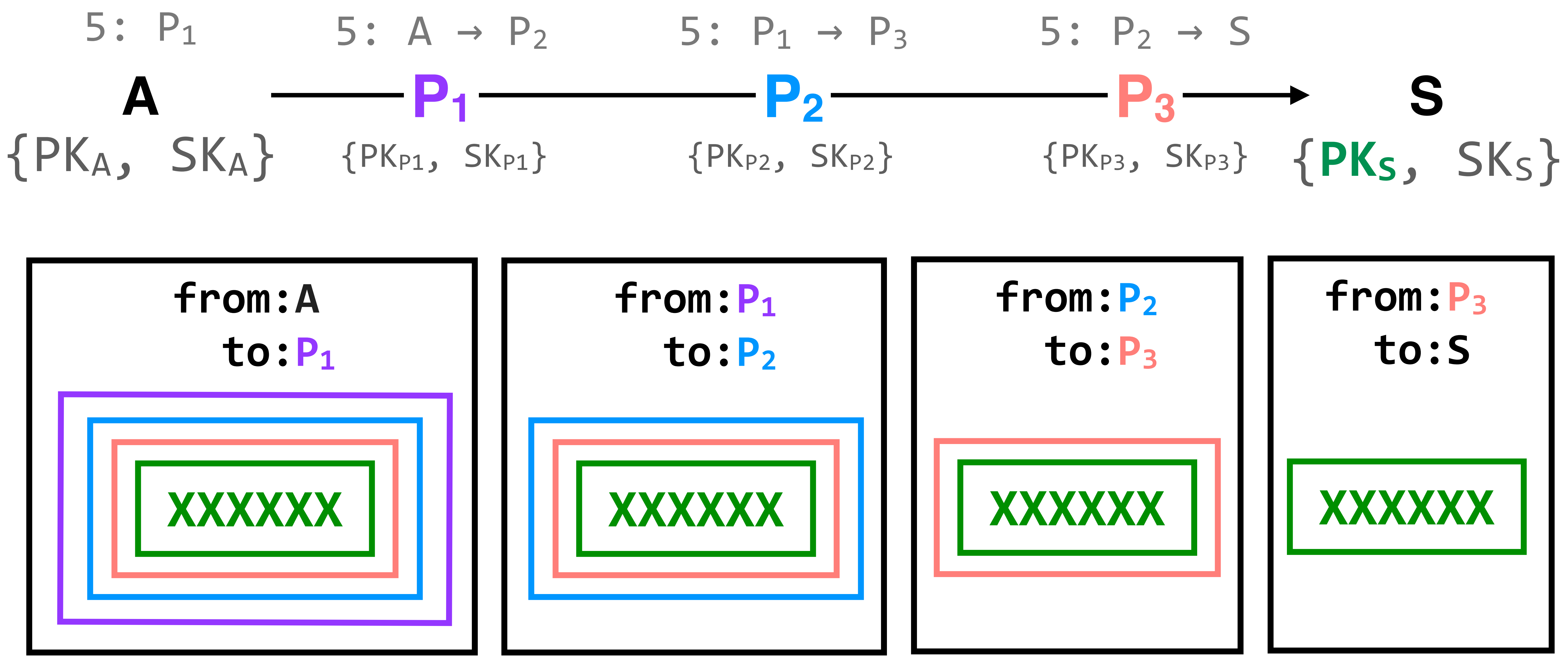
public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$\text{encrypt}(PK_x, m) = c$$
$$\text{decrypt}(SK_x, c) = m$$

things to avoid

- no packet should say "from: A; to: S"
- no entity in the network should receive a packet from A and send it directly to S
- no entity in the network should keep *state* that links A to S
- data should not appear the same across packets

5 is the (random) "circuit ID". a proxy may be involved in more than one circuit, so it needs a way to differentiate



onion routing adds layers of encryption that nodes on the path can strip off as the packet traverses the network

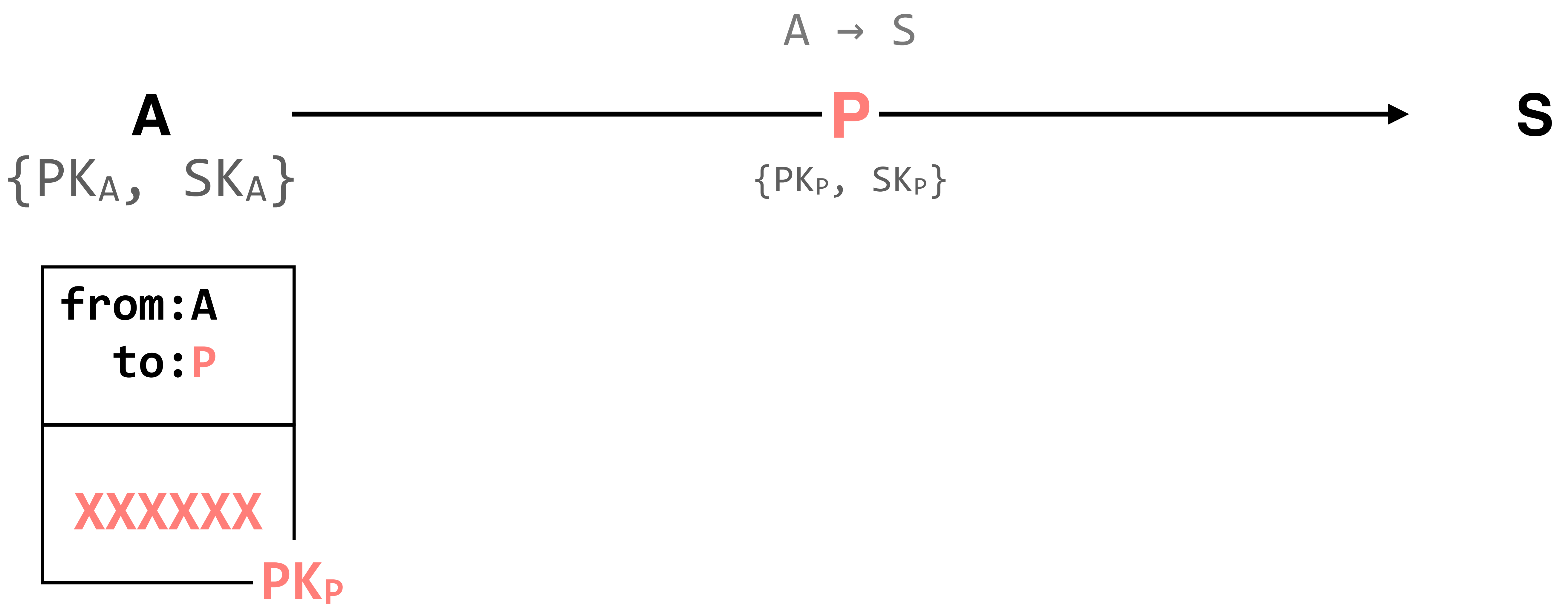
in practice, **tor** uses public-key cryptography to securely exchange **symmetric keys** between A and each node in the circuit, and the layers of encryption use those symmetric keys; this is what allow traffic to travel in both directions

policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

```
encrypt(PKx, m) = c
decrypt(SKx, c) = m
```



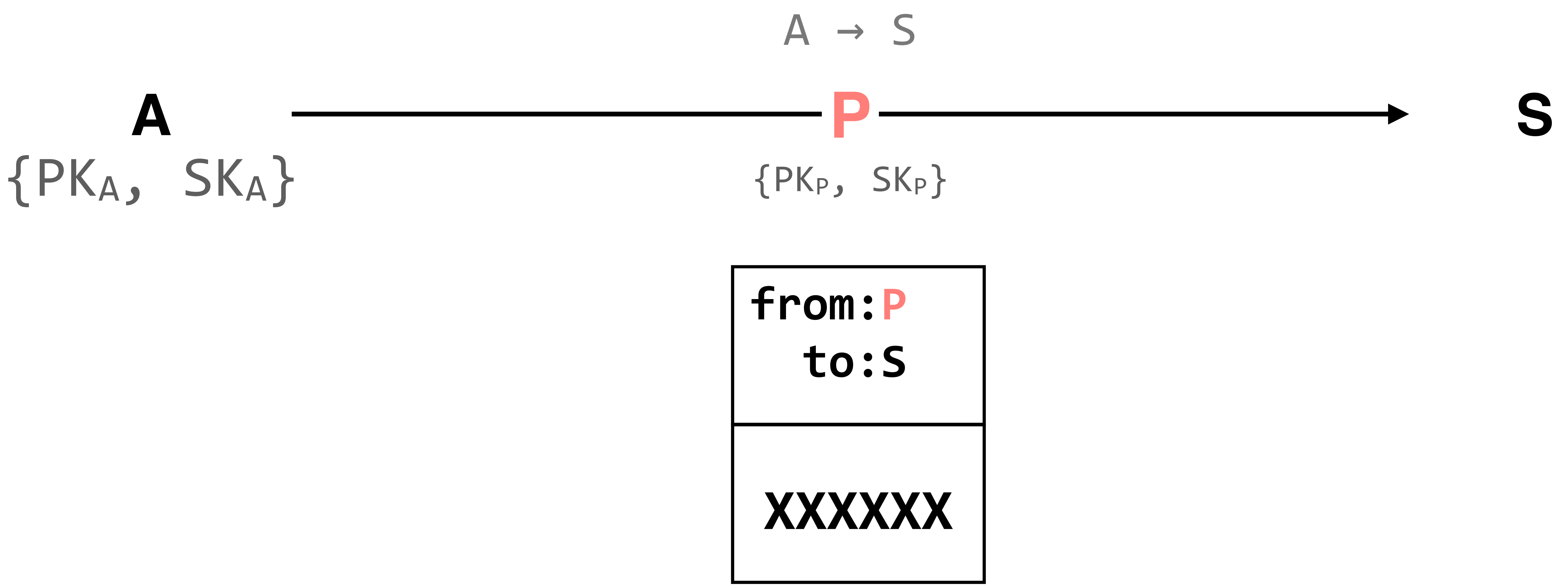
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

`encrypt(PK_x, m) = c`

`decrypt(SK_x, c) = m`



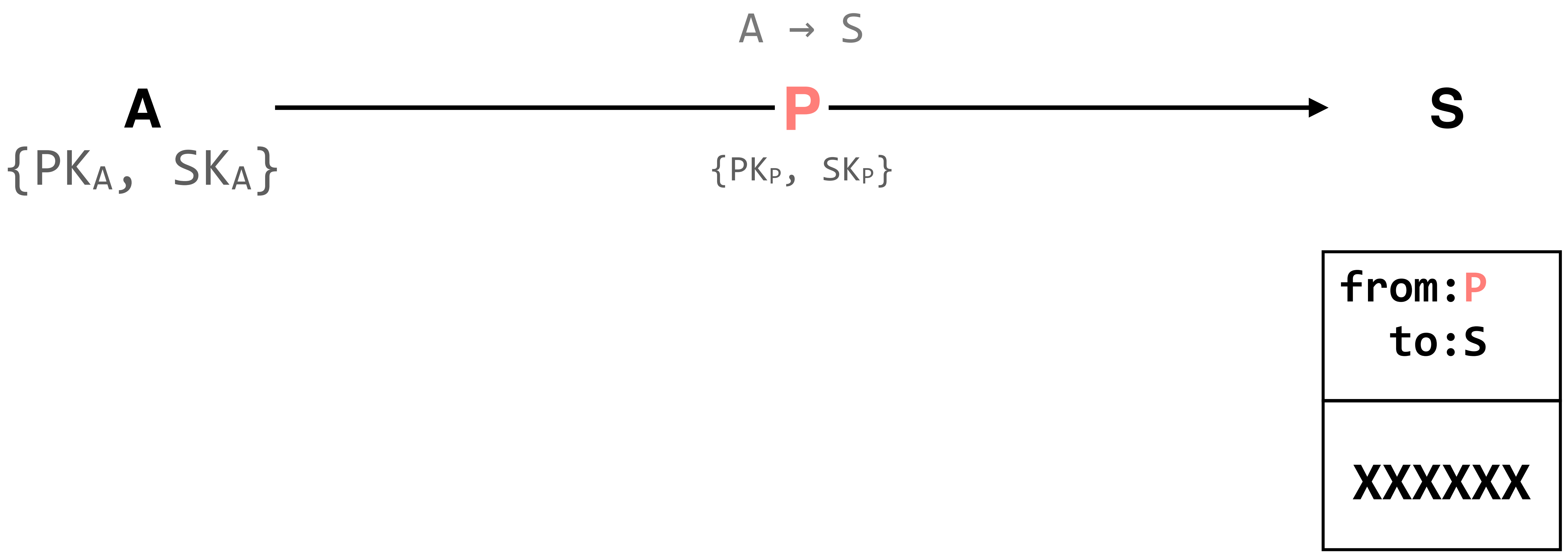
policy: provide **anonymity** (only the client should know that they're communicating with the server)

threat model: adversary is on the path between the client and the server

public-key cryptography: a message to **x** is encrypted with **x**'s public key; only **x**'s secret key can decrypt the message

$$\text{encrypt}(PK_x, m) = c$$

$$\text{decrypt}(SK_x, c) = m$$



assuming you trust the proxy, this type of service can be useful if you care about confidentiality on a local network

what we've shown here is a simplified version of some of the functionality you get when you use a VPN

tor provides some level of anonymity for users, preventing adversaries from linking a sender to its receiver

much like when we discussed certificate authorities, there are interesting questions about who should run tor. how do we trust that the relay nodes are behaving as they should?

there are still ways to attack tor, namely by **correlating traffic** from various points in the network

as system designers, it's important to think about what traffic you're sending over the network to clients, and whether that traffic can be sent in a more secure way (and what the trade-offs would be)

a larger takeaway here is that a secure channel alone only provides confidentiality and integrity of the message data; **packet headers can reveal information** that may be sensitive in certain contexts