In search of CurveSwap: Measuring elliptic curve implementations in the wild

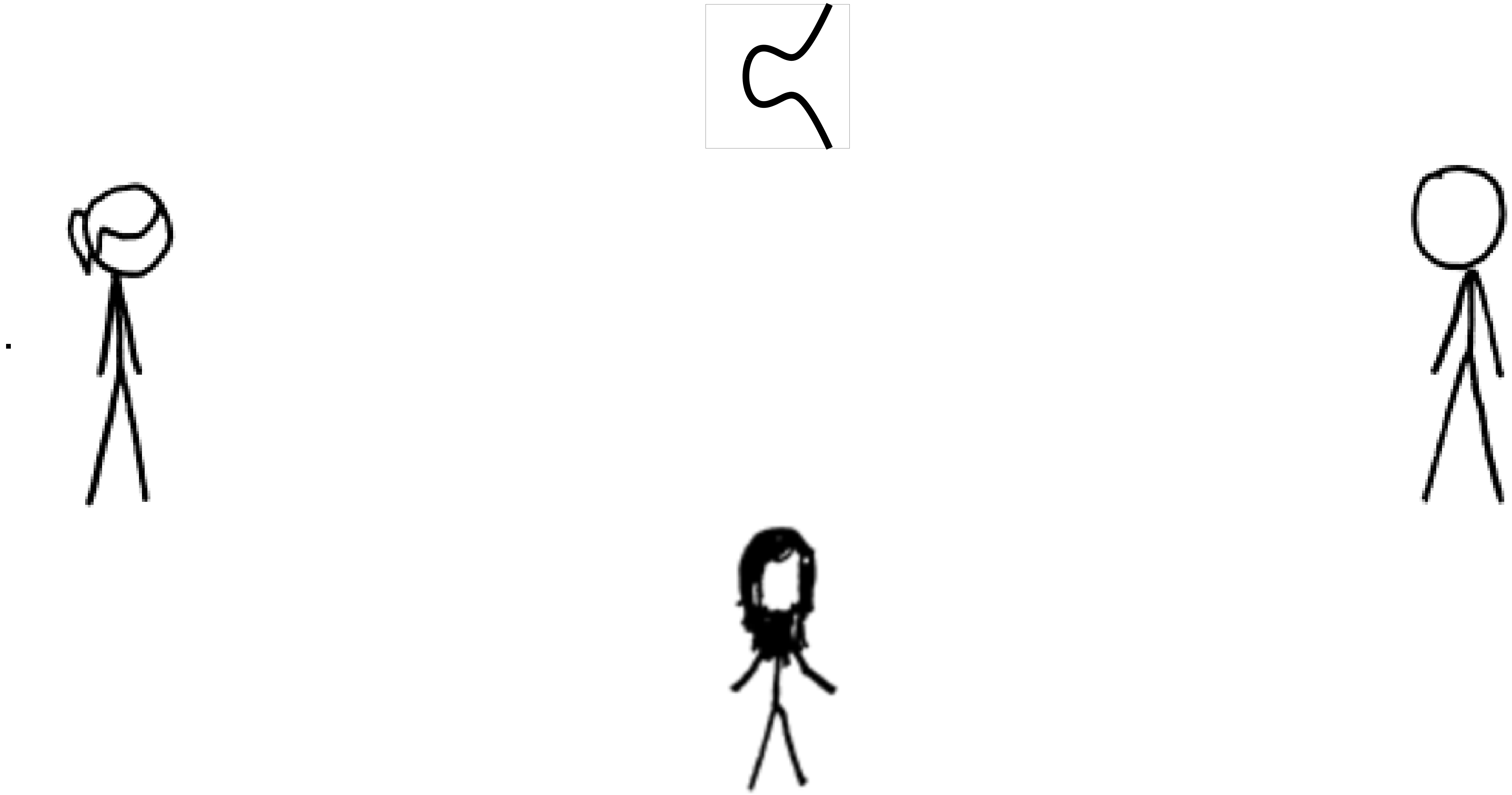
Luke Valenta*, Nick Sullivan†, Antonio Sanso‡, Nadia Heninger*

*University of Pennsylvania, †Cloudflare, ‡Adobe

December 6th, 2018

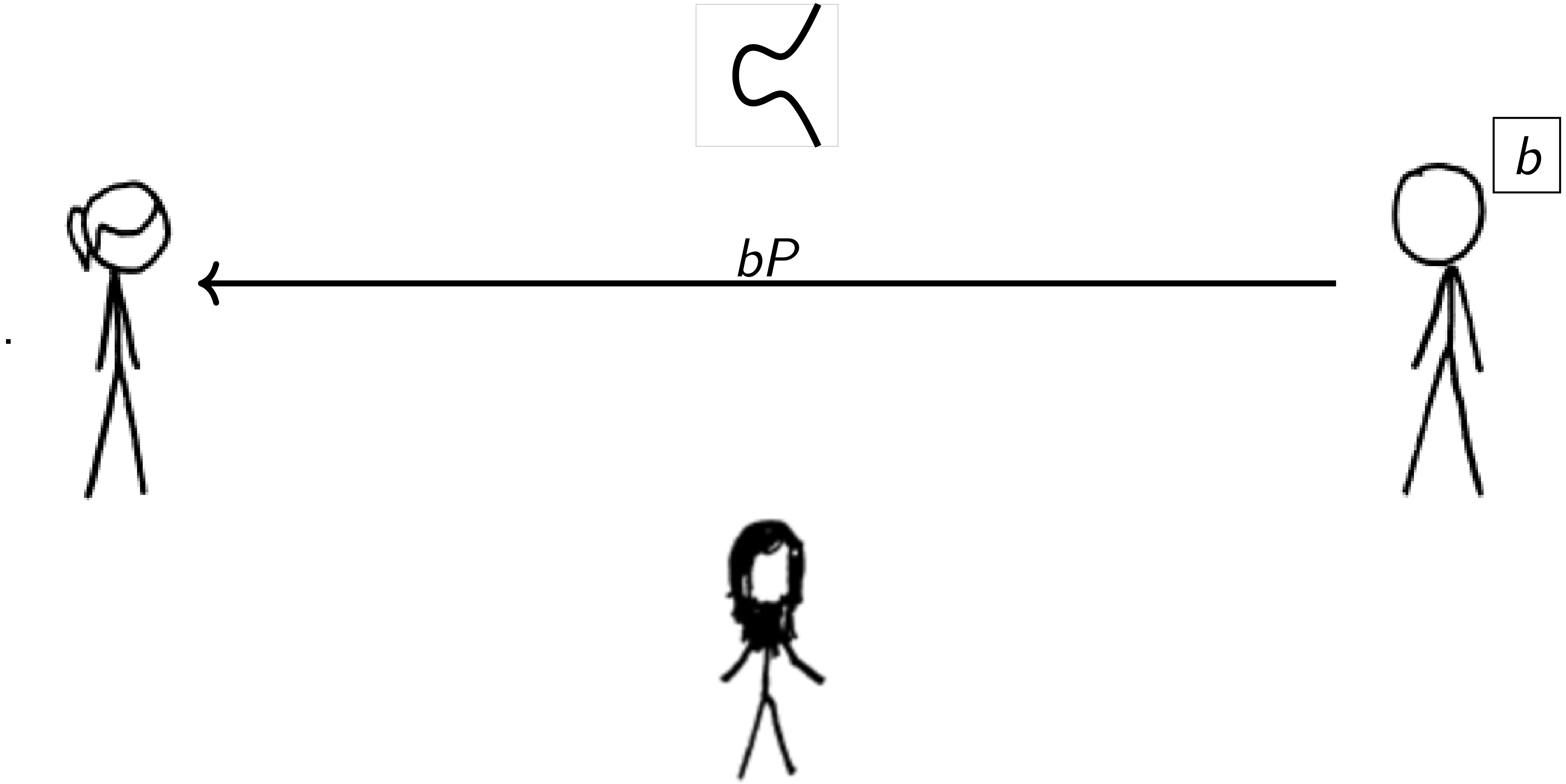
Elliptic Curve Diffie-Hellman (ECDH)

client "Alice", server "Bob", eavesdropper "Eve"



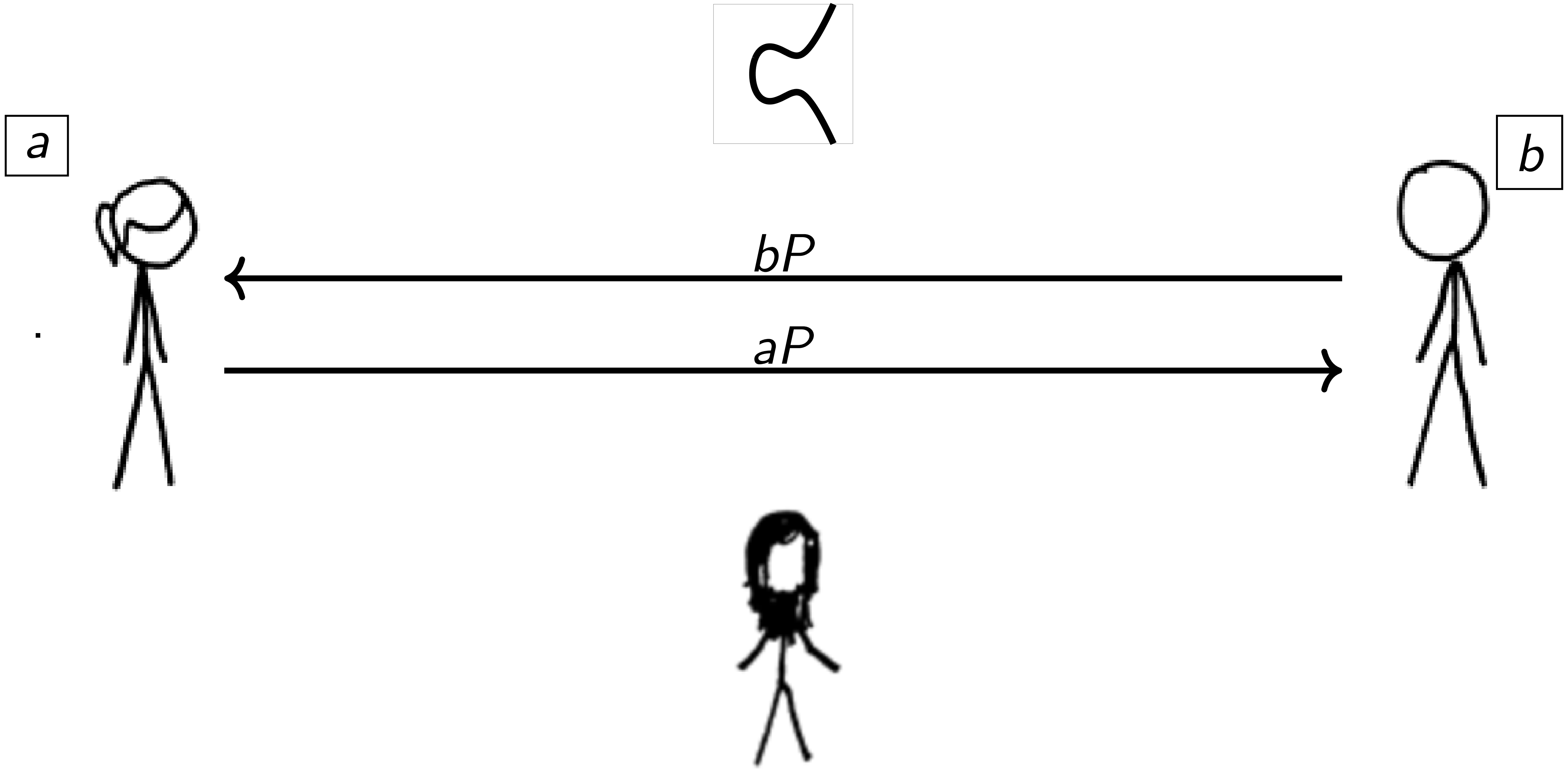
Elliptic Curve Diffie-Hellman (ECDH)

client “Alice”, server “Bob”, eavesdropper “Eve”



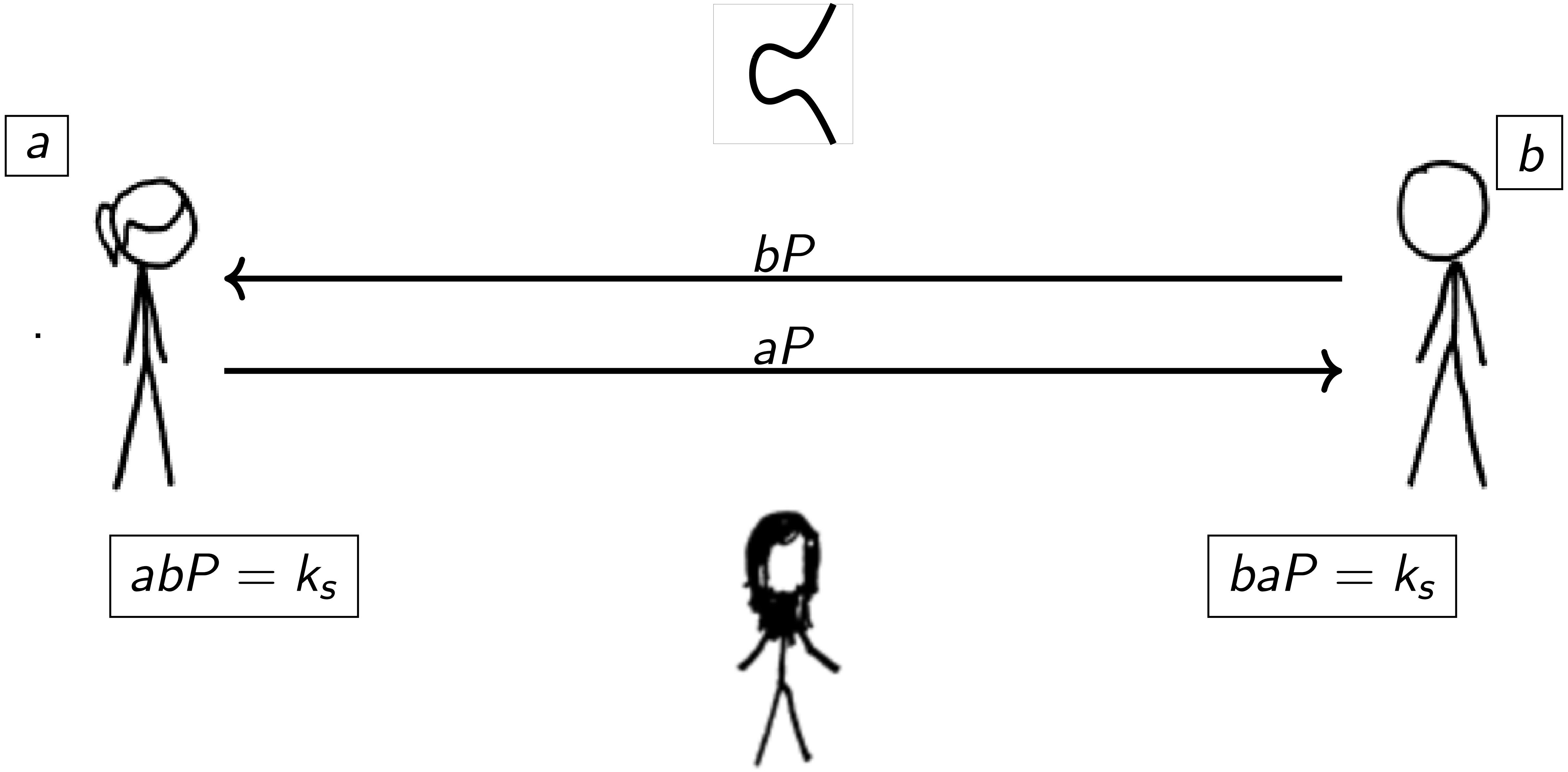
Elliptic Curve Diffie-Hellman (ECDH)

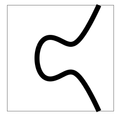
client “Alice”, server “Bob”, eavesdropper “Eve”



Elliptic Curve Diffie-Hellman (ECDH)

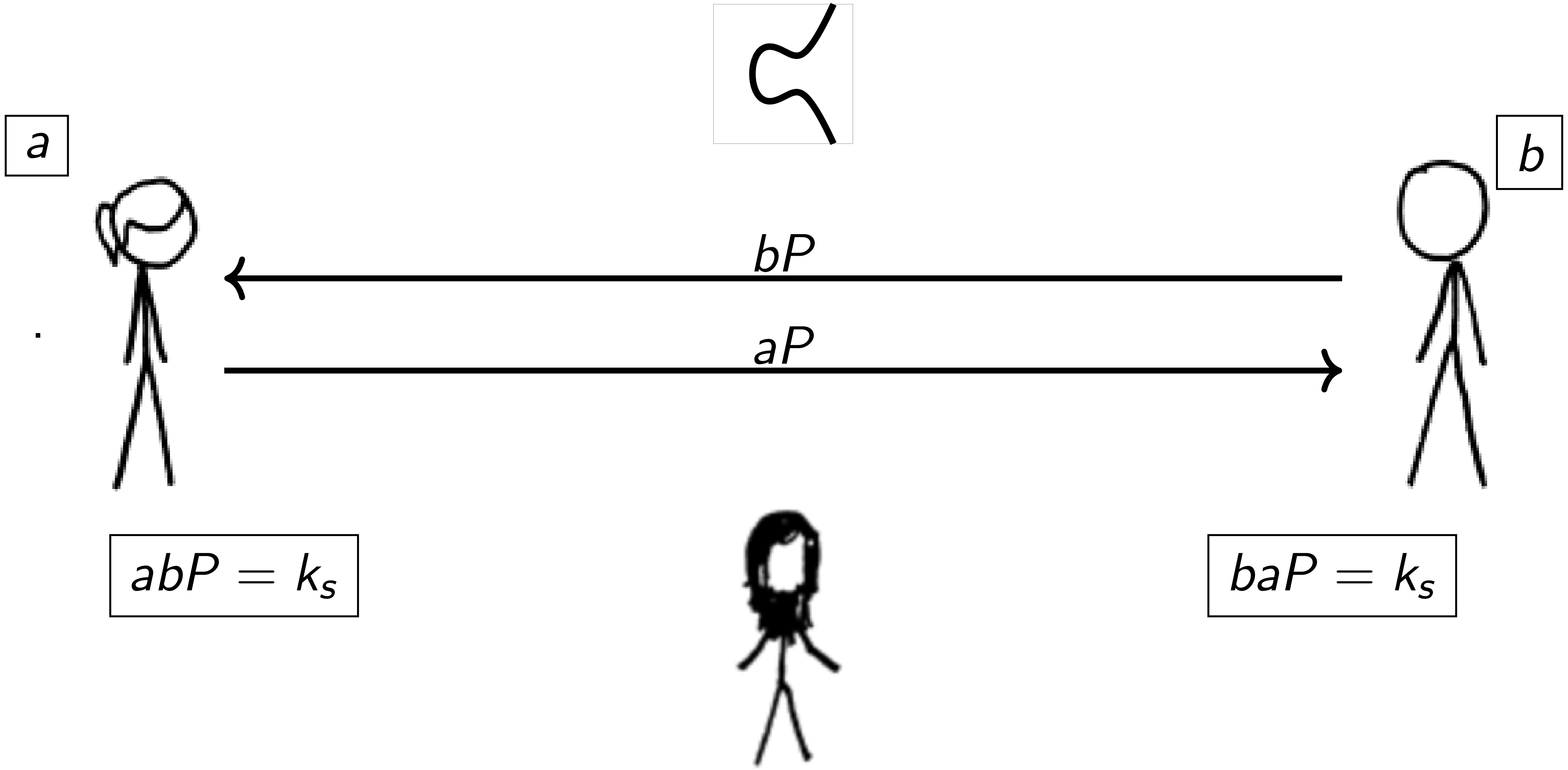
client "Alice", server "Bob", eavesdropper "Eve"

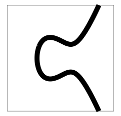


CDH assumption: Given aP, bP, and , Eve should not learn k_s

Elliptic Curve Diffie-Hellman (ECDH)

client “Alice”, server “Bob”, eavesdropper “Eve”

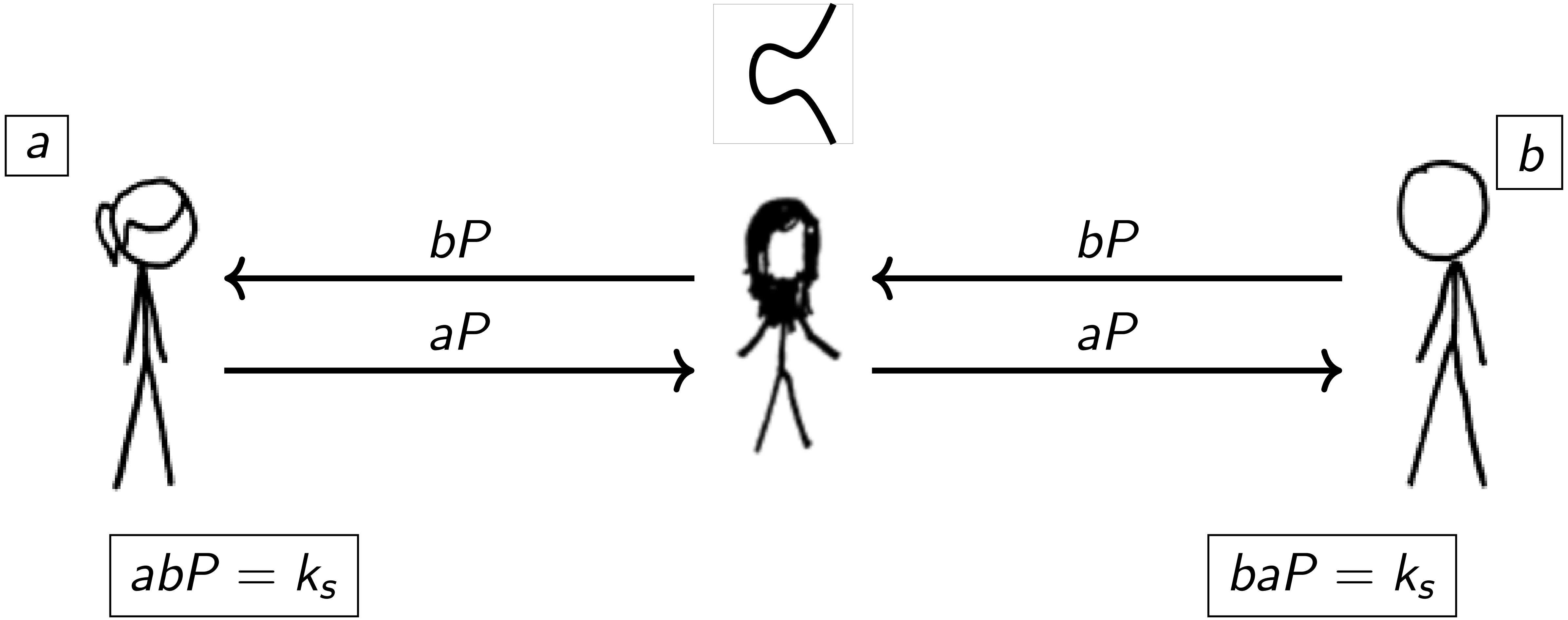


CDH assumption: Given aP, bP, and , Eve should not learn k_s

…but this is vulnerable to MitM attack

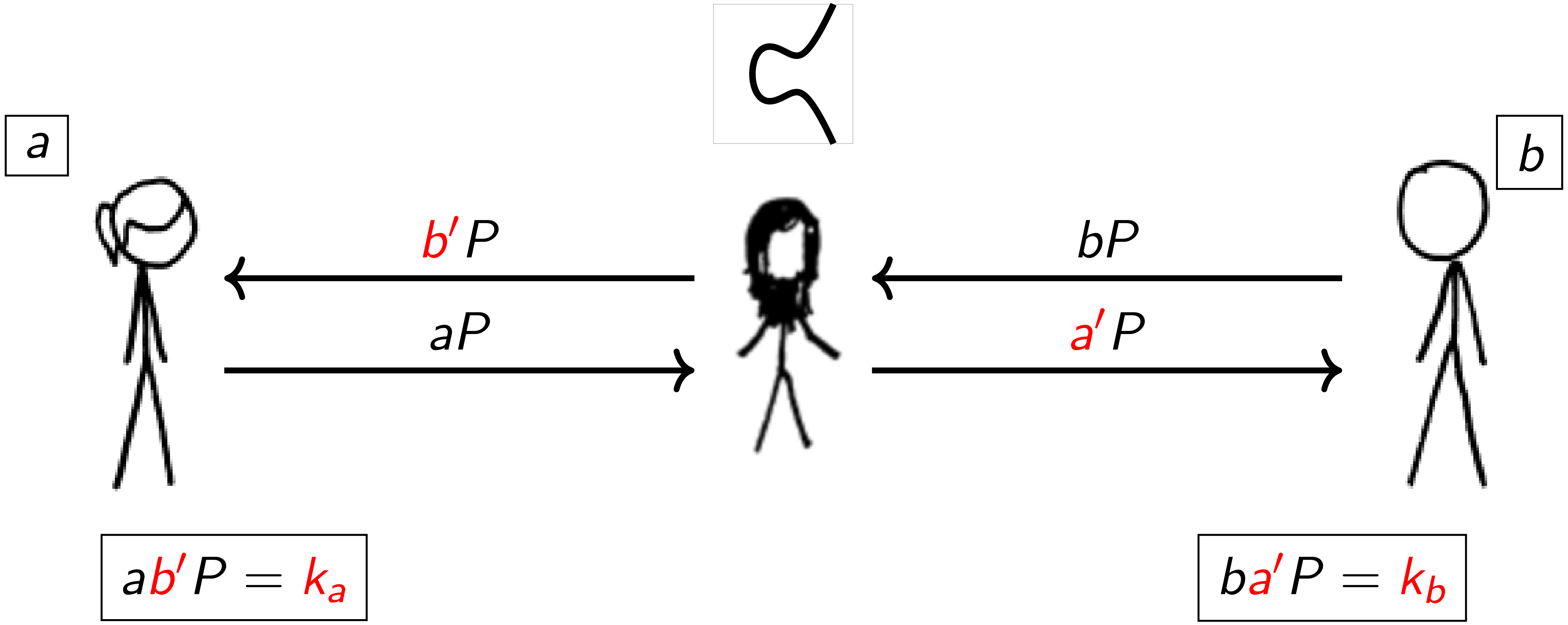
Elliptic Curve Diffie-Hellman (ECDH)

client “Alice”, server “Bob”, man in the middle “Mallory”



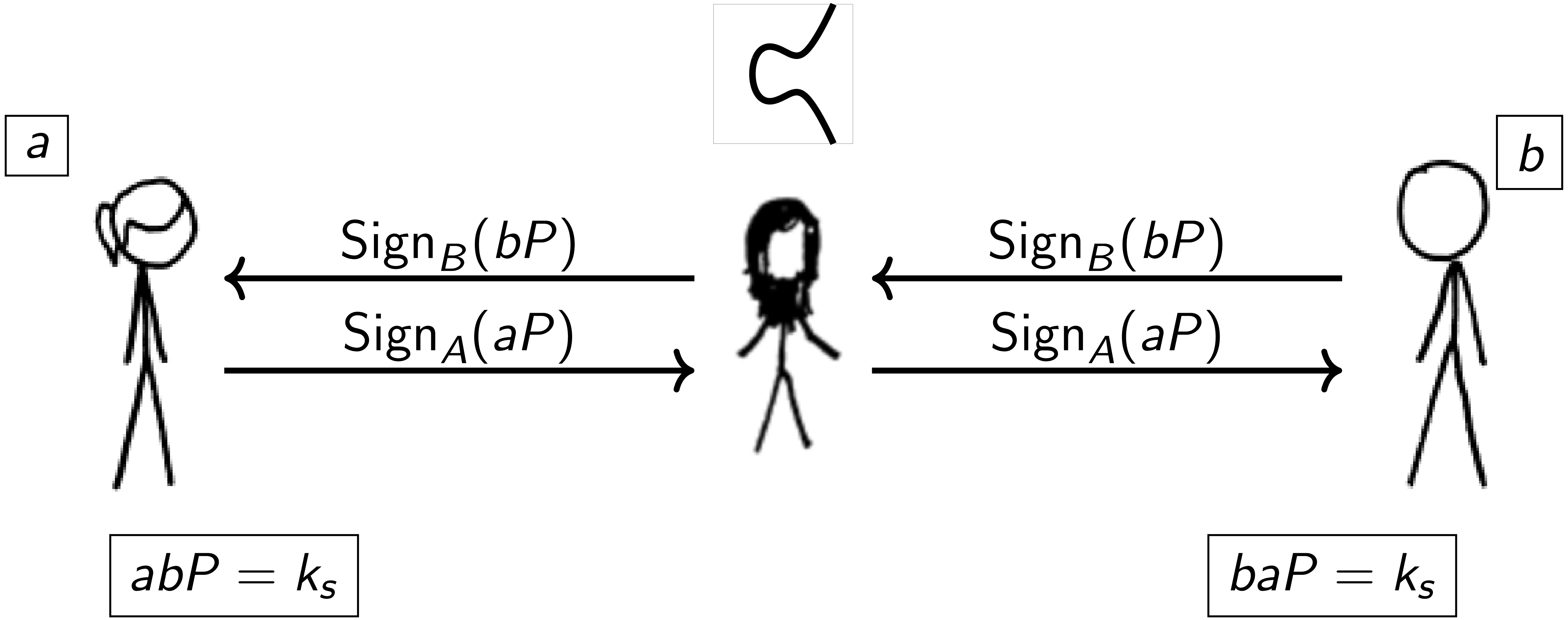
Elliptic Curve Diffie-Hellman (ECDH)

client "Alice", server "Bob", man in the middle "Mallory"



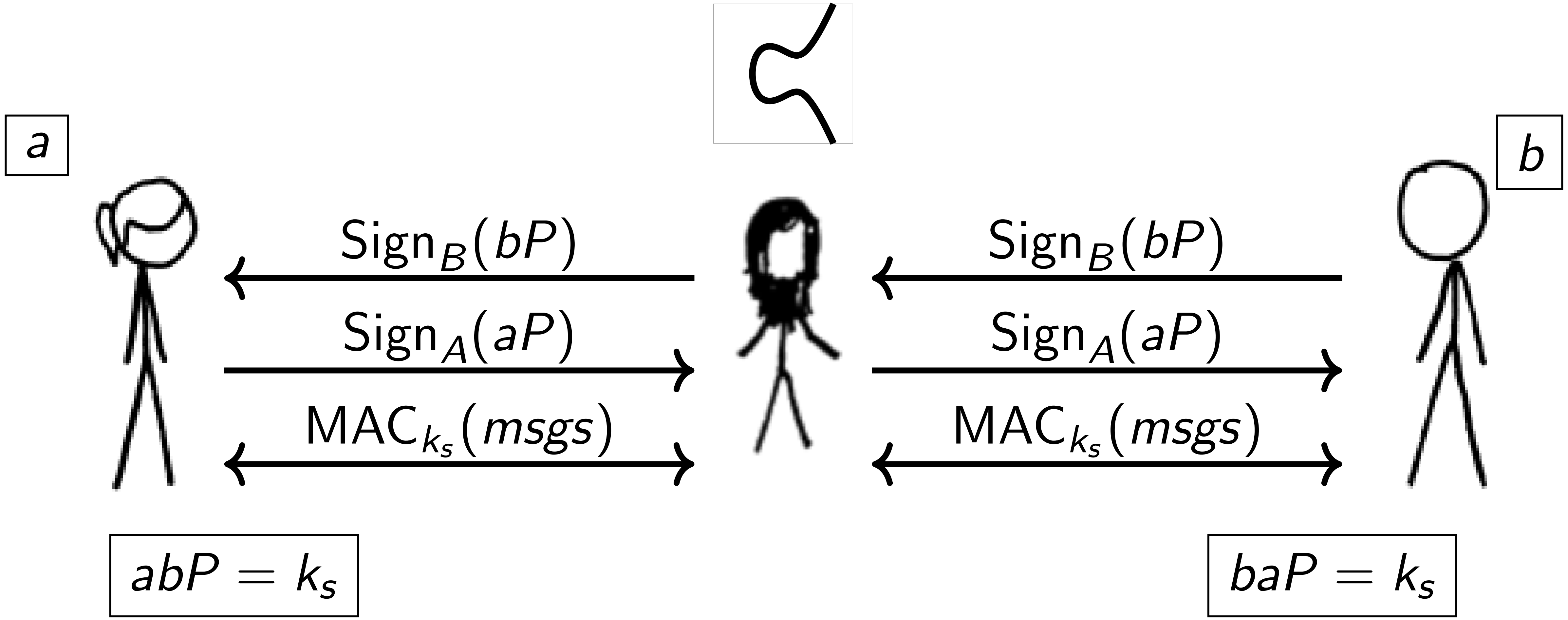
Elliptic Curve Diffie-Hellman (ECDH) w/ authentication

client "Alice", server "Bob", man in the middle "Mallory"



Elliptic Curve Diffie-Hellman (ECDH) w/ authentication

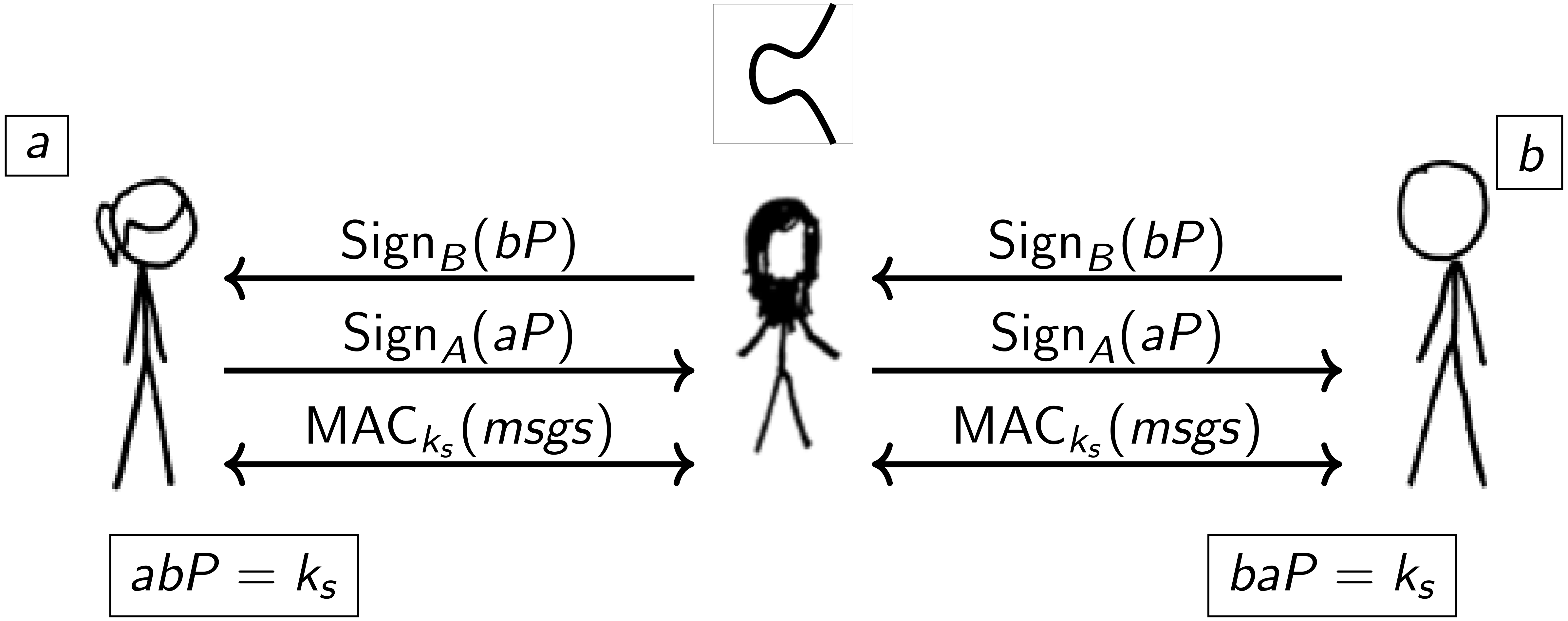
client "Alice", server "Bob", man in the middle "Mallory"



Signatures/MAC prevent naïve MitM

Elliptic Curve Diffie-Hellman (ECDH) w/ authentication

client "Alice", server "Bob", man in the middle "Mallory"

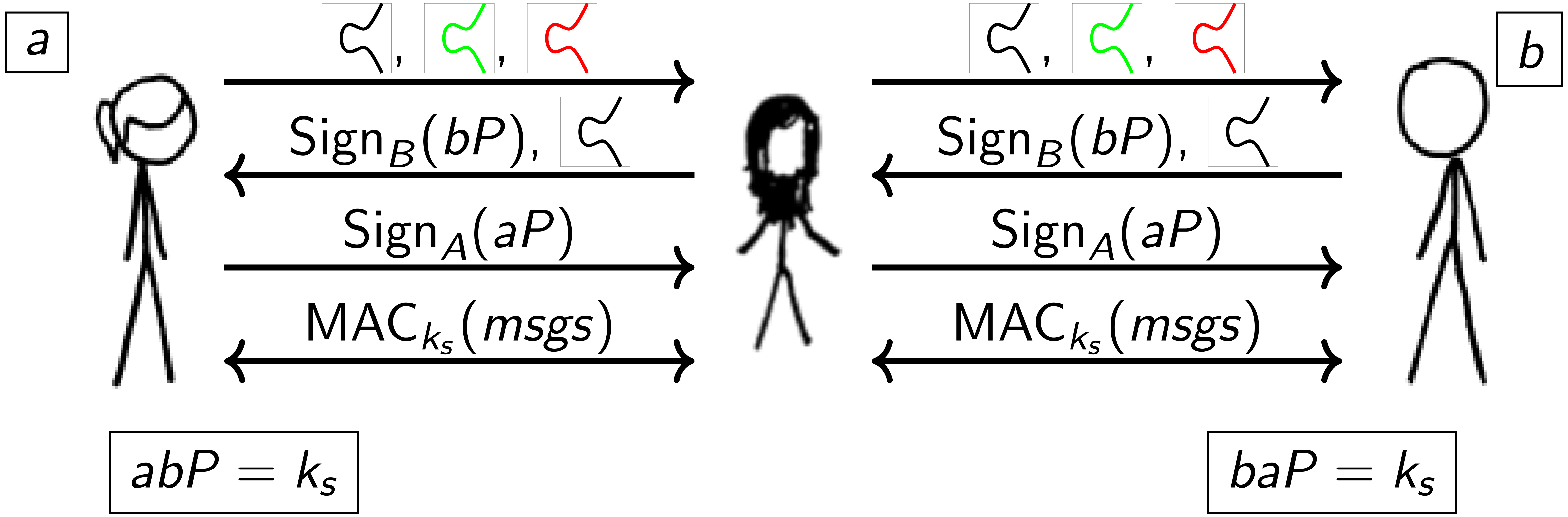


Signatures/MAC prevent naïve MitM

. . . but how do Alice and Bob decide on the curve?

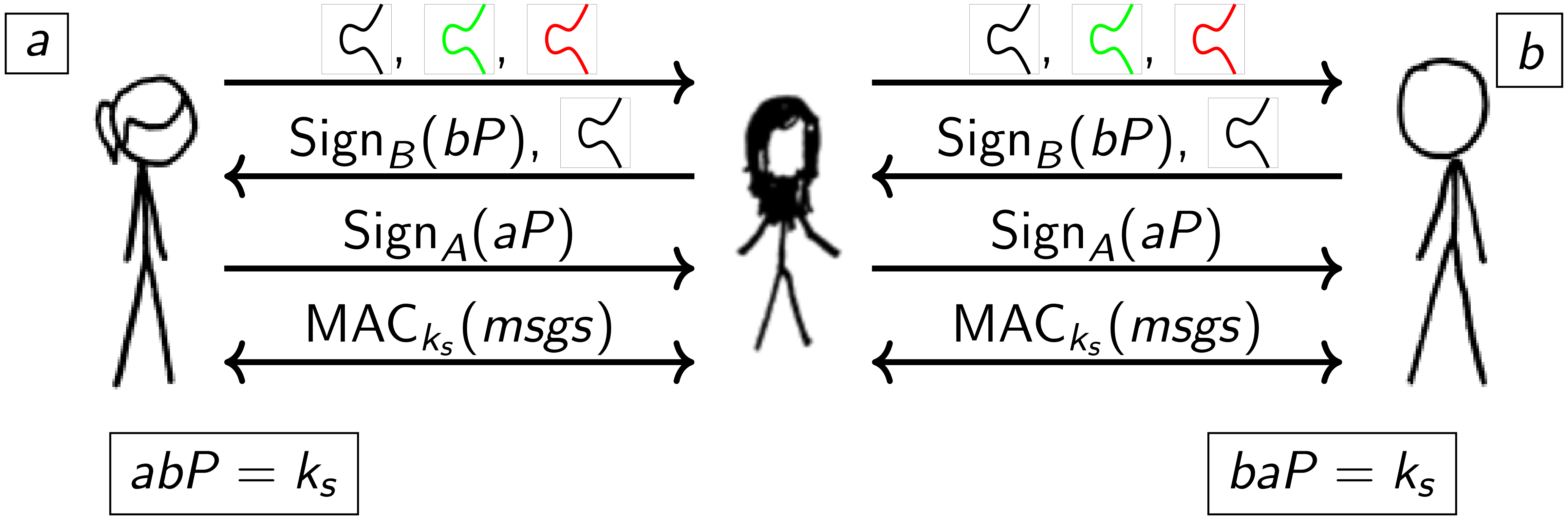
Elliptic Curve Diffie-Hellman (ECDH) w/ curve negotiation

client "Alice", server "Bob", man in the middle "Mallory"



Elliptic Curve Diffie-Hellman (ECDH) w/ curve negotiation

client "Alice", server "Bob", man in the middle "Mallory"



Curve negotiation is not authenticated in TLS 1.2

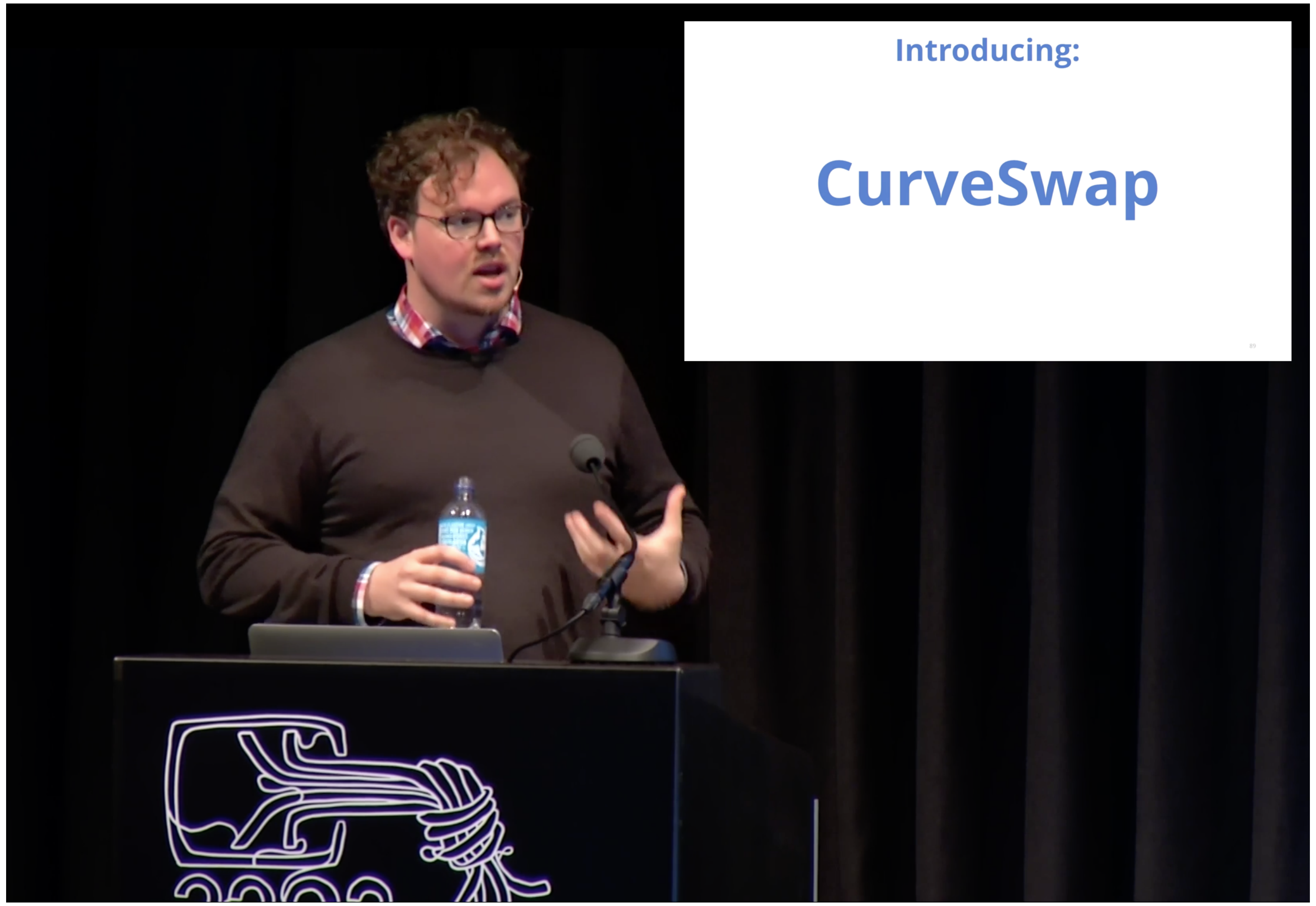
CurveSwap

Nick Sullivan at 32C3 (2015):

"TLS supports a ton of crazy elliptic curves"

"what if you did a downgrade attack on that?"

"take the supported curves, and swap it with the smallest weakest curves supported by both parties"

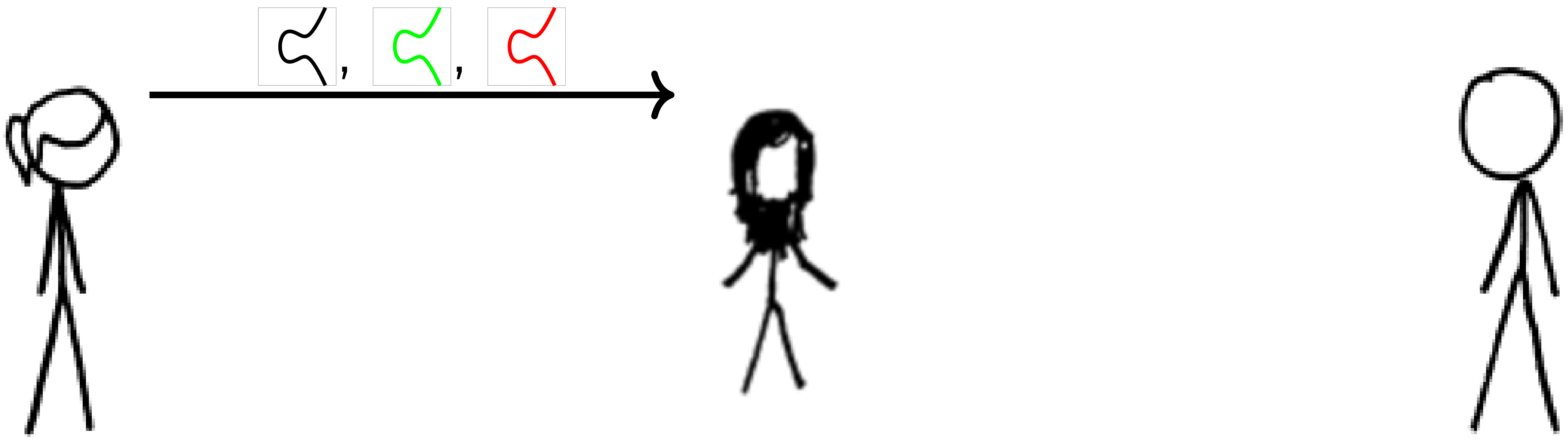


CurveSwap attack

client "Alice", server "Bob", man in the middle "Mallory"

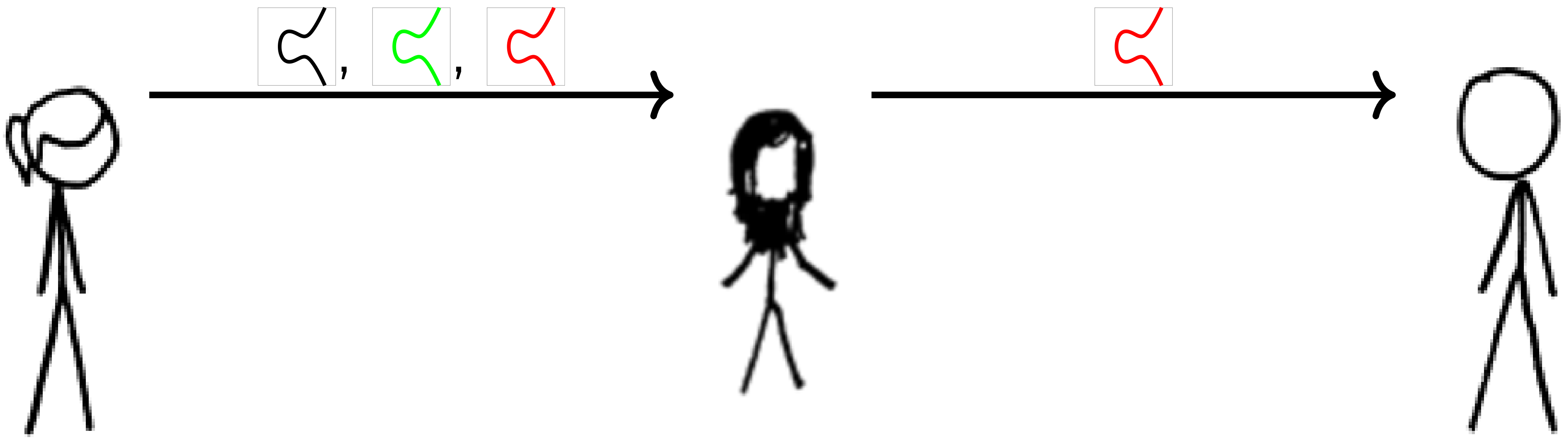
CurveSwap attack

client “Alice”, server “Bob”, man in the middle “Mallory”



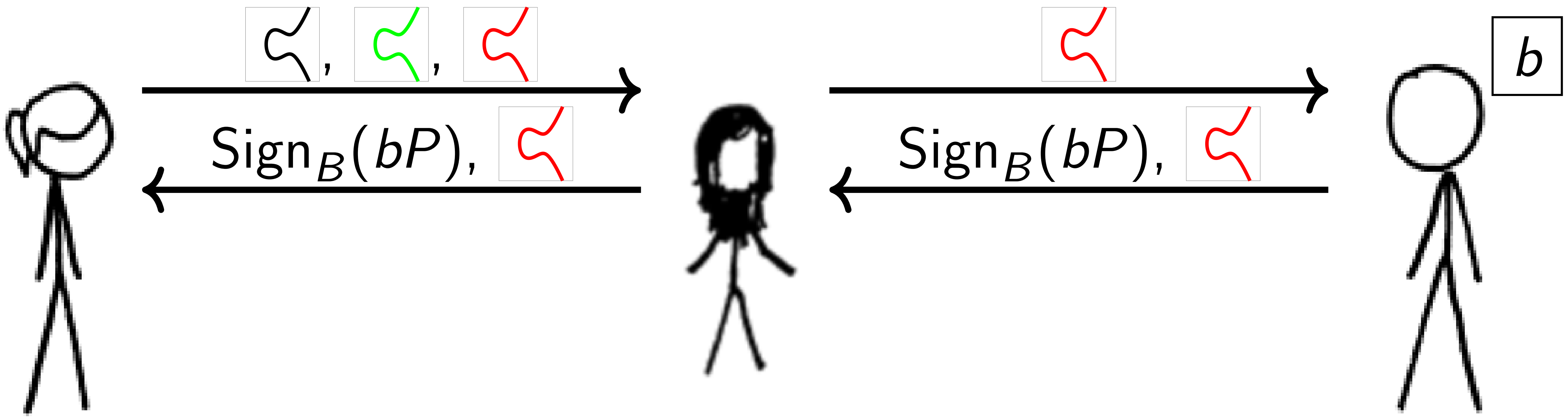
CurveSwap attack

client “Alice”, server “Bob”, man in the middle “Mallory”



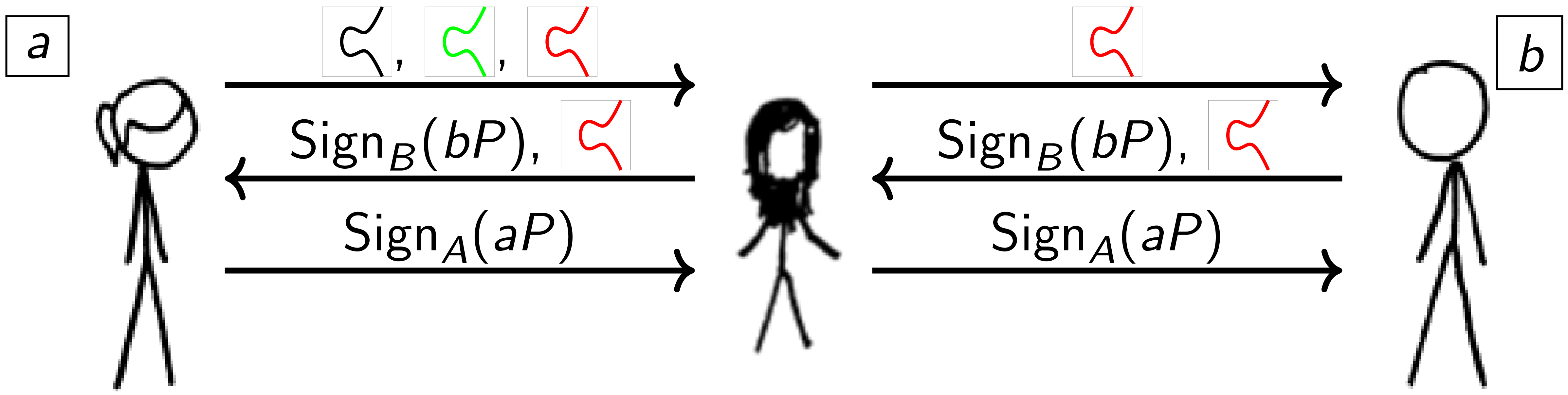
CurveSwap attack

client “Alice”, server “Bob”, man in the middle “Mallory”



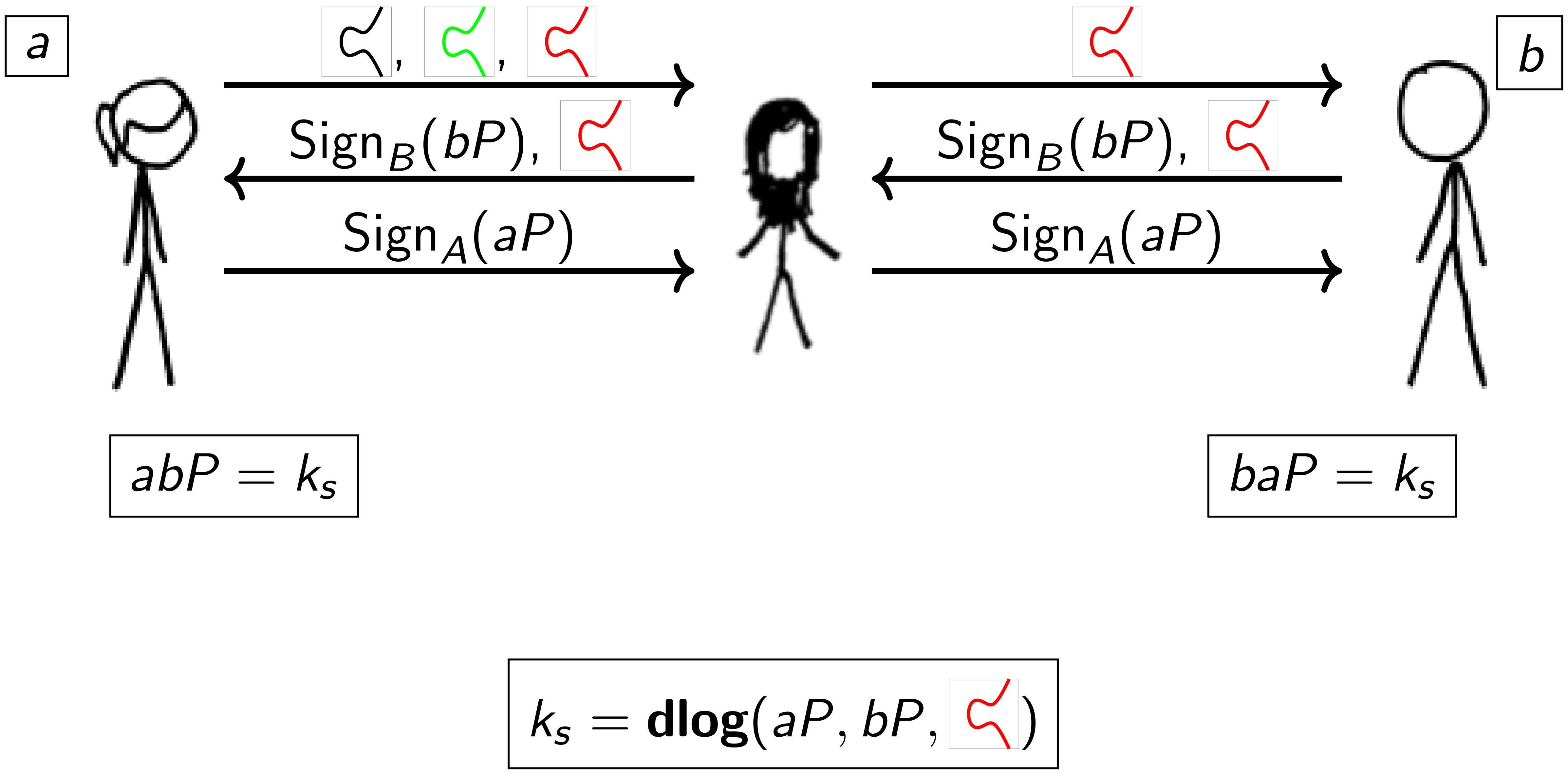
CurveSwap attack

client "Alice", server "Bob", man in the middle "Mallory"



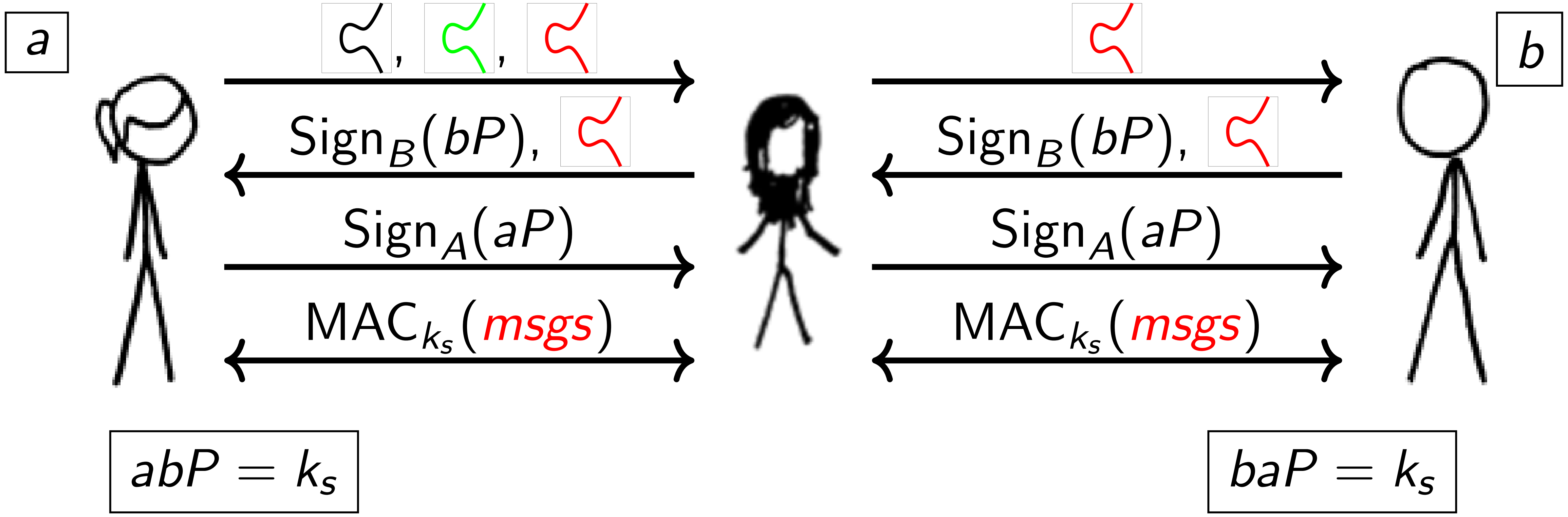
CurveSwap attack

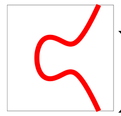
client "Alice", server "Bob", man in the middle "Mallory"



CurveSwap attack

client “Alice”, server “Bob”, man in the middle “Mallory”



$k_s = \textbf{dlog}(aP, bP,$  $)$

MAC only depends on k_s

This work

Evaluate feasibility of CurveSwap downgrade attack

- Requires breaking ECDH online for some supported curve

This work

Evaluate feasibility of CurveSwap downgrade attack

- Requires breaking ECDH online for some supported curve

Look at ECDH in TLS, SSH, IPsec (IKE), JWE

Measure elliptic curve usage in hosts and implementations

This work

Evaluate feasibility of CurveSwap downgrade attack

- Requires breaking ECDH online for some supported curve

Look at ECDH in TLS, SSH, IPsec (IKE), JWE

Measure elliptic curve usage in hosts and implementations

Punch line: we find many weaknesses in elliptic curve implementations, but nobody vulnerable to CurveSwap

Scan measurements

Fast internet scanning lets us study behavior of publicly accessib
hosts.

Scan measurements

Fast internet scanning lets us study behavior of publicly accessib
hosts.

Curve support across protocols varies widely

	Total	ECDHE	secp224r1	secp256r1	x25519
HTTPS	41.0M	28.8M	2.8%	86.9%	2.6%
SSH	14.5M	7.9M	0.0%	97.8%	77.2%
IKEv1	1.1M	215.4K	66.8%	98.3%	0.0%
IKEv2	1.2M	101.1K	4.1%	97.1%	0.0%

Scan measurements

Fast internet scanning lets us study behavior of publicly accessib
hosts.

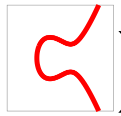
Curve support across protocols varies widely

	Total	**ECDHE**	**secp224r1**	**secp256r1**	**x25519**
HTTPS	41.0M	28.8M	2.8%	86.9%	2.6%
SSH	14.5M	7.9M	0.0%	97.8%	77.2%
IKEv1	1.1M	215.4K	66.8%	98.3%	0.0%
IKEv2	1.2M	101.1K	4.1%	97.1%	0.0%

8.5M HTTPS servers chose `secp256r1`, `secp384r1`, or `secp521r1`, even when not offered by the client.

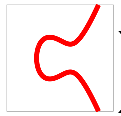
Breaking Elliptic Curve Diffie-Hellman

CurveSwap requires breaking ECDH for some supported curve

$k_s = \mathbf{dlog}(aP, bP,$  $)$

Breaking Elliptic Curve Diffie-Hellman

CurveSwap requires breaking ECDH for some supported curve

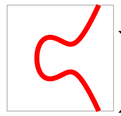
$k_s = \textbf{dlog}(aP, bP,$  $)$

Known attack vectors

- Solve the discrete logarithm on weak curves
- Invalid point attacks

Breaking Elliptic Curve Diffie-Hellman

CurveSwap requires breaking ECDH for some supported curve

$k_s = \mathbf{dlog}(aP, bP,$  $)$

Known attack vectors

- Solve the discrete logarithm on weak curves
- Invalid point attacks

Need server to reuse key for multiple connections

- Common optimization to reduce server load

Do servers reuse keys?

Scanned each host on public IPv4 Internet twice in rapid succession with `secp256r1`, a popular curve.

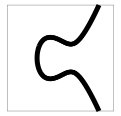
Do servers reuse keys?

Scanned each host on public IPv4 Internet twice in rapid succession with `secp256r1`, a popular curve.

Of the TLS hosts supporting `secp256r1`:

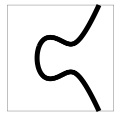
- 5.5M (22%) reused keys at least once
- 640K (2.6%) used the same key as another host

Solve the discrete logarithm on weak curves

ECDLP: Given  and bP, compute b

Best known attack runs in $\mathcal{O}(\sqrt{n})$ for curve with n points

Solve the discrete logarithm on weak curves

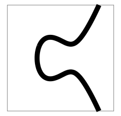
ECDLP: Given  and bP, compute b

Best known attack runs in $\mathcal{O}(\sqrt{n})$ for curve with n points

TLS supports a ton of weak elliptic curves

- `secp160r1` has 80-bit security
- Bitcoin network computes 2^{80} hashes every 11 hours

Solve the discrete logarithm on weak curves

ECDLP: Given  and bP, compute b

Best known attack runs in $\mathcal{O}(\sqrt{n})$ for curve with n points

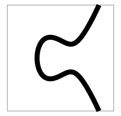
TLS supports a ton of weak elliptic curves

- `secp160r1` has 80-bit security
- Bitcoin network computes 2^{80} hashes every 11 hours

Out of 4M **client** hellos:

- sampled from Cloudflare
- 682.6K (16.3%) support `secp160r1`

Solve the discrete logarithm on weak curves

ECDLP: Given  and bP, compute b

Best known attack runs in $\mathcal{O}(\sqrt{n})$ for curve with n points

TLS supports a ton of weak elliptic curves

- `secp160r1` has 80-bit security
- Bitcoin network computes 2^{80} hashes every 11 hours

Out of 4M **client** hellos:

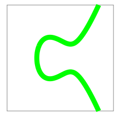
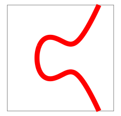
- sampled from Cloudflare
- 682.6K (16.3%) support `secp160r1`

Out of 41M **servers** from scans:

- 276.2K (0.67%) support `secp160r1`
- 8.1K (2.9%) also reused keys
- only 2 reused after 25 hours

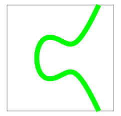
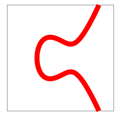
Invalid point attacks

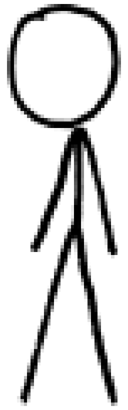
Some implementations are “curve blind”

Lack the validation checks to differentiate between  and 

Invalid point attacks

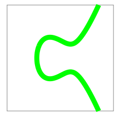
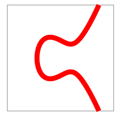
Some implementations are “curve blind”

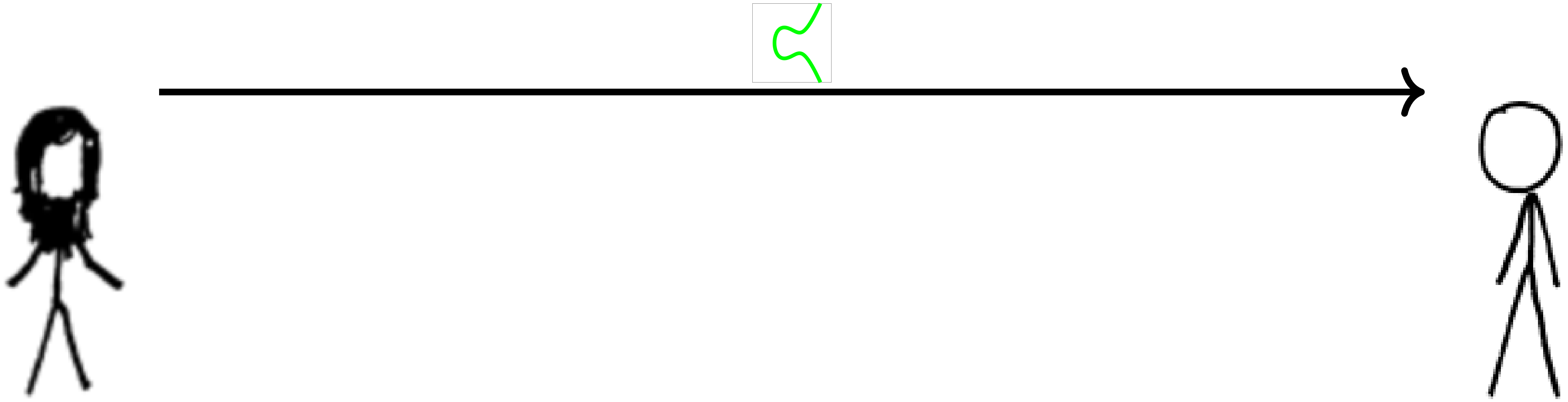
Lack the validation checks to differentiate between  and 



Invalid point attacks

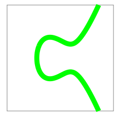
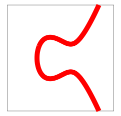
Some implementations are "curve blind"

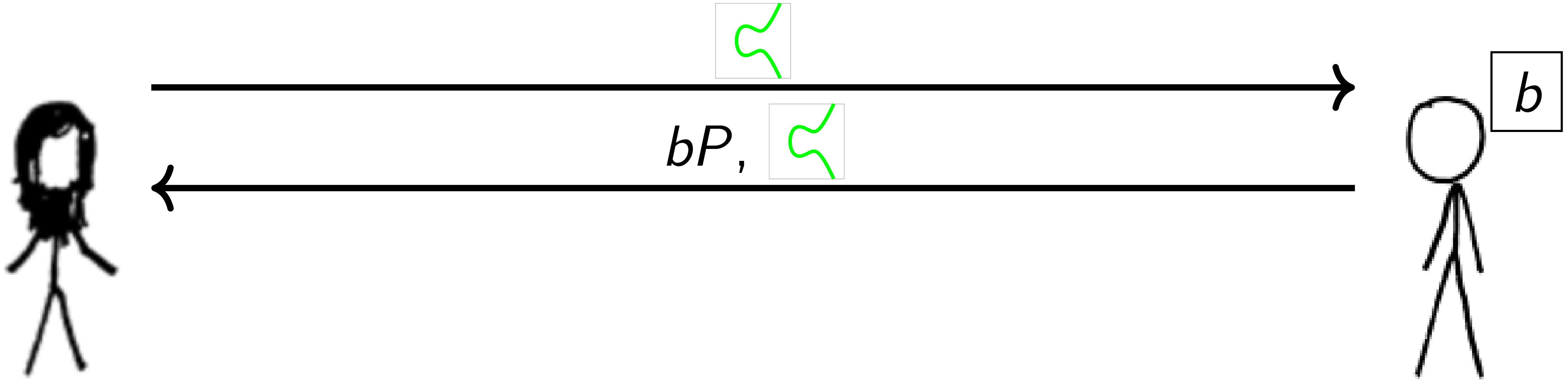
Lack the validation checks to differentiate between  and 



Invalid point attacks

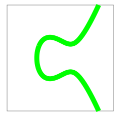
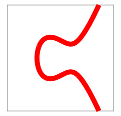
Some implementations are “curve blind”

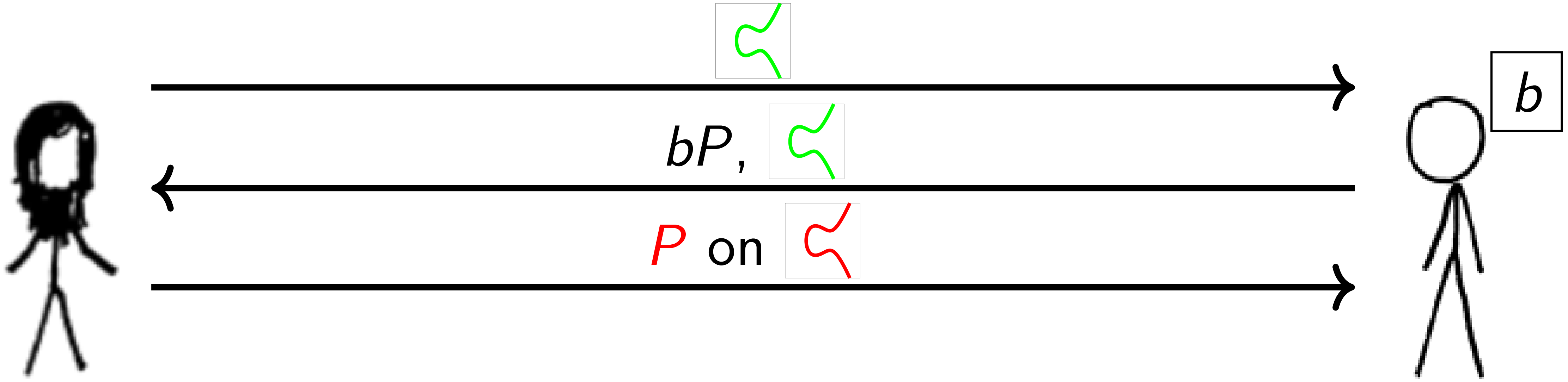
Lack the validation checks to differentiate between  and 



Invalid point attacks

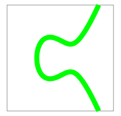
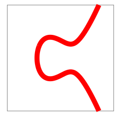
Some implementations are "curve blind"

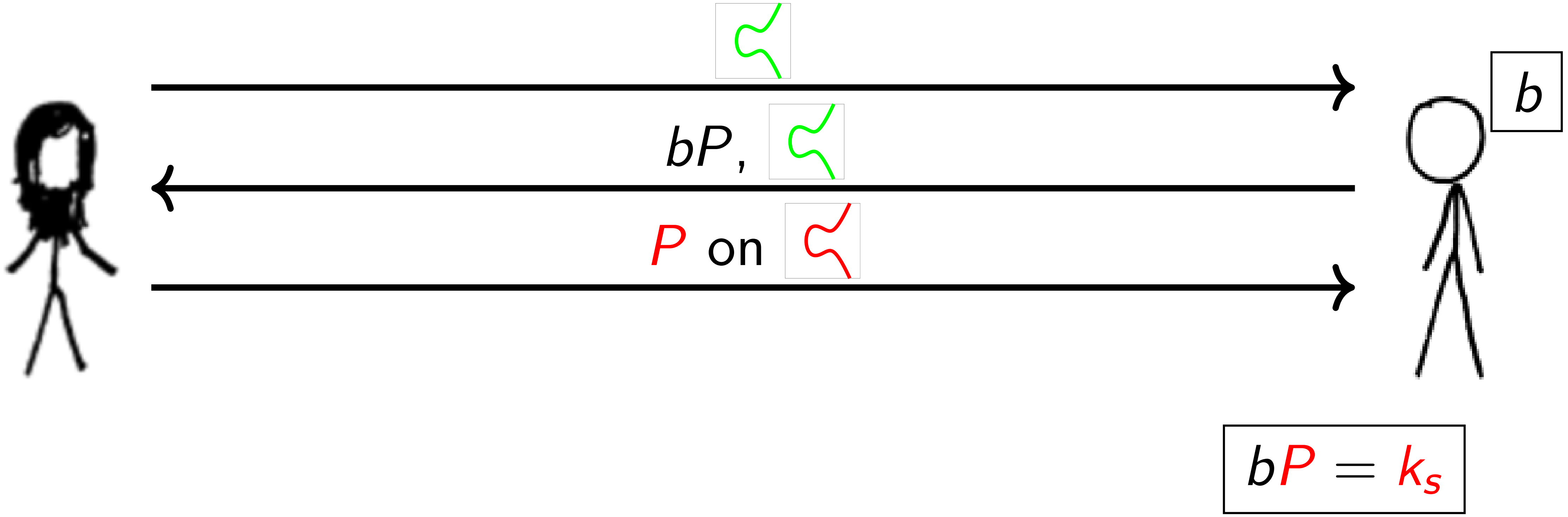
Lack the validation checks to differentiate between  and 



Invalid point attacks

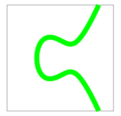
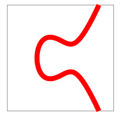
Some implementations are "curve blind"

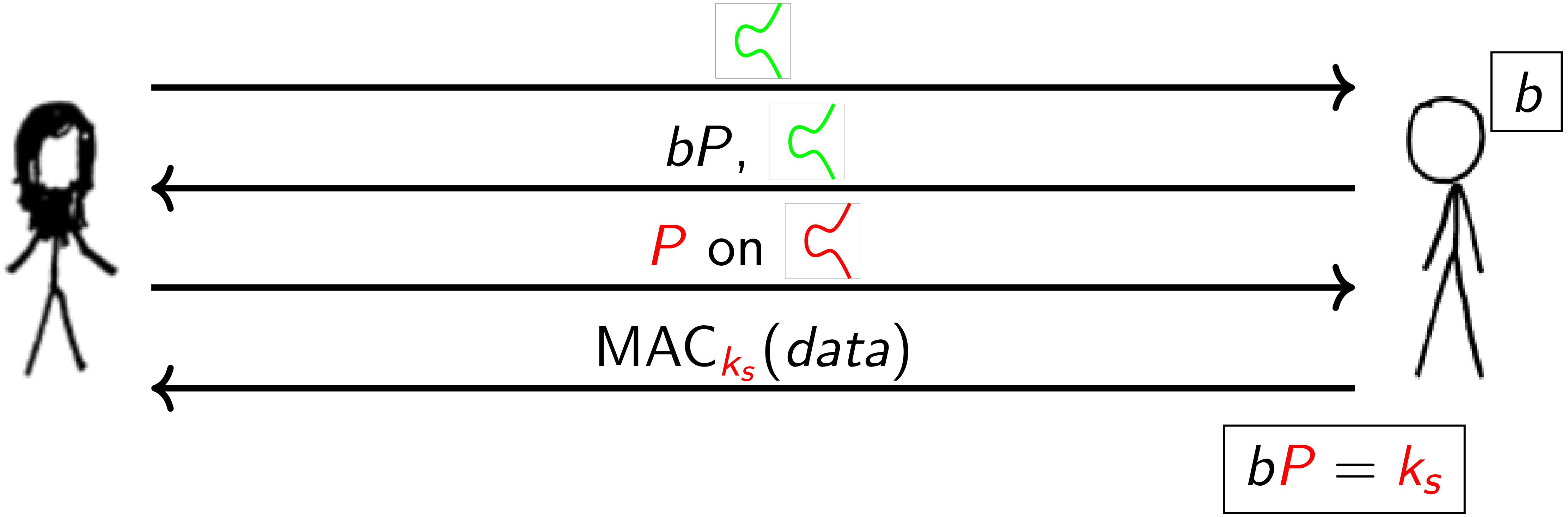
Lack the validation checks to differentiate between  and 



Invalid point attacks

Some implementations are “curve blind”

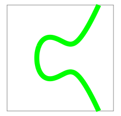
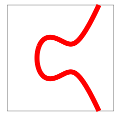
Lack the validation checks to differentiate between  and 

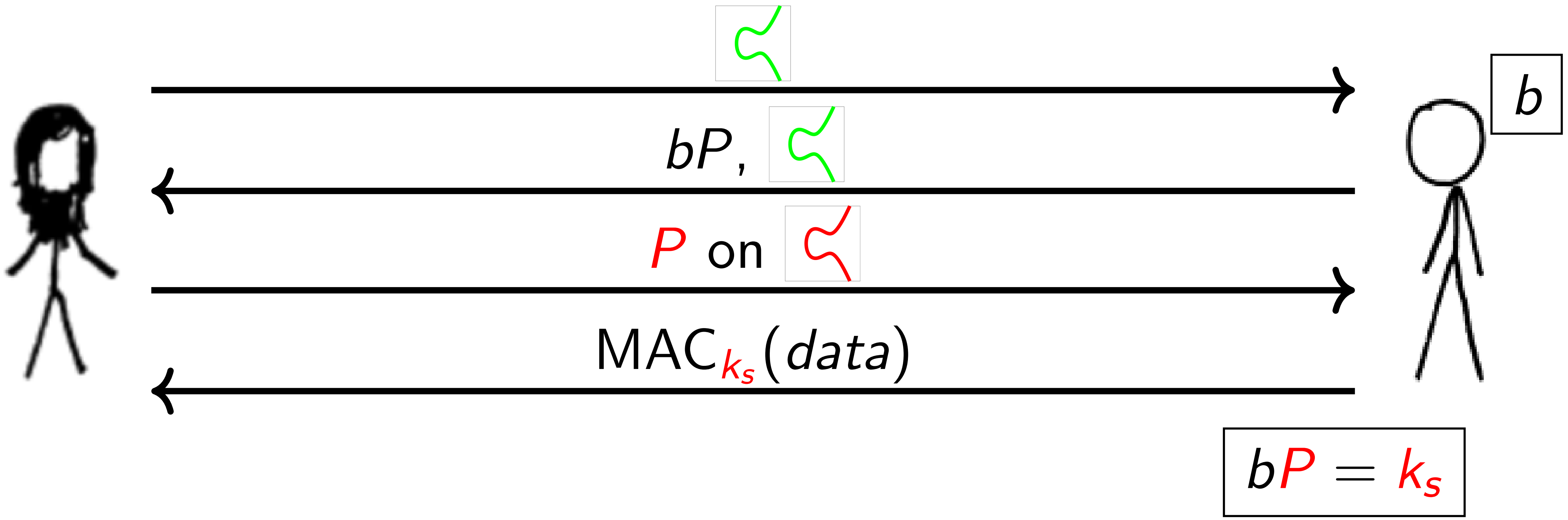


$\textbf{break}(\mathrm{MAC}_{k_s}(data)) \implies$ learn some bits of b

Invalid point attacks

Some implementations are “curve blind”

Lack the validation checks to differentiate between  and 



break(MAC$_{k_s}$($data$)) $\implies$ learn some bits of b

Repeat many times $\implies$ find b using Chinese Remainder Theorem

Countermeasures

The countermeasures against these attacks are well known, and built into ~~all~~ most ECDH standards:

RFC 4492 (TLS): “The server retrieves the client’s ephemeral ECDH public key from the ClientKeyExchange message and checks that it is on the same elliptic curve as the server’s ECDH key.”

RFC 5656 (SSH): “All elliptic curve public keys MUST be validated after they are received”

RFC 6989 (IKEv2): “A receiving peer MUST check that its peer’s public key value is valid”

RFC 7516, 7518 (JWE): . . . no warning?

Do libraries validate public keys?

Many TLS libraries don't validate for ECDH: [JSS ESORICS '15]

Similar for FFDH in TLS, SSH, IPsec: [VASCFHHH NDSS '16]

Do libraries validate public keys?

Many TLS libraries don't validate for ECDH: [JSS ESORICS '15]

Similar for FFDH in TLS, SSH, IPsec: [VASCFHHH NDSS '16]

Many JWE libraries don't validate:

Library	**Vulnerable**
jose4j	Yes
Nimbus JOSE+JWT	Yes
Apache CXF	No
go-jose	Yes
jose2go	Yes
node-jose	Yes

Do hosts validate public keys?

Scanning methodology: test for two types of curve blindness

- send order-5 point on *invalid* curve related to `secp256r1`
- send order-5 point on *twist* of `secp256r1`

Do hosts validate public keys?

Scanning methodology: test for two types of curve blindness

- send order-5 point on *invalid* curve related to `secp256r1`
- send order-5 point on *twist* of `secp256r1`

Protocol	**Accept**	**Accept + Reuse Keys**
HTTPS	188.7K (0.7%)	0 (0.0%)
SSH*	4.1K (0.1%)	0 (0.0%)
IKEv1*	530 (0.2%)	0 (0.0%)
IKEv2*	4.1K (4.0%)	0 (0.0%)

* Overestimates due to scanning limitations

Scans from November 2016

Modern advancements in ECC

"New" DJB curves: `Curve25519`, `Curve41417`, `Curve448`

- Montgomery/twisted Edwards curves
- By design, no twist or invalid curve attacks
- `Curve25519` supported by 77.2% of SSH, 2.6% of HTTPS
- TLS 1.3 includes `Curve25519` and `Curve448`

Black Hat Sound Bytes

Standards writers:

- Easy for developers to skip validation checks
- Minimize complexity of curve support
- Downgrade protection is essential in protocol design

Black Hat Sound Bytes

Standards writers:

- Easy for developers to skip validation checks
- Minimize complexity of curve support
- Downgrade protection is essential in protocol design

Software developers:

- Cryptography is hard, but it is easy to prevent known attacks
- Cryptographic validation should be part of your test suite (`https://github.com/google/wycheproof`)

Black Hat Sound Bytes

Standards writers:

- Easy for developers to skip validation checks
- Minimize complexity of curve support
- Downgrade protection is essential in protocol design

Software developers:

- Cryptography is hard, but it is easy to prevent known attacks
- Cryptographic validation should be part of your test suite (`https://github.com/google/wycheproof`)

Academic researchers:

- Internet scanning is an effective "black box" measurement tool
- "Negative" results can and should be published

Black Hat Sound Bytes

Standards writers:

- Easy for developers to skip validation checks
- Minimize complexity of curve support
- Downgrade protection is essential in protocol design

Software developers:

- Cryptography is hard, but it is easy to prevent known attacks
- Cryptographic validation should be part of your test suite (`https://github.com/google/wycheproof`)

Academic researchers:

- Internet scanning is an effective "black box" measurement tool
- "Negative" results can and should be published

Questions?

References

In search of CurveSwap: measuring elliptic curve implementations in the wild Luke Valenta, Nick Sullivan, Antonio Sanso, Nadia Heninger. *EuroS&P 2018.* `https://eprint.iacr.org/2018/298`

Practical invalid curve attacks on TLS-ECDH Tibor Jager, Jörg Schwenk, Juraj Somorovsky. *ESORICS 2015.*

Measuring small subgroup attacks against Diffie-Hellman Luke Valenta, David Adrian, Antonio Sanso, Shaanan Cohney, Joshua Fried, Marcella Hastings, J. Alex Halderman, Nadia Heninger. *NDSS 2016.*

Alice, Bob, and Eve images from Randall Munroe (XKCD)