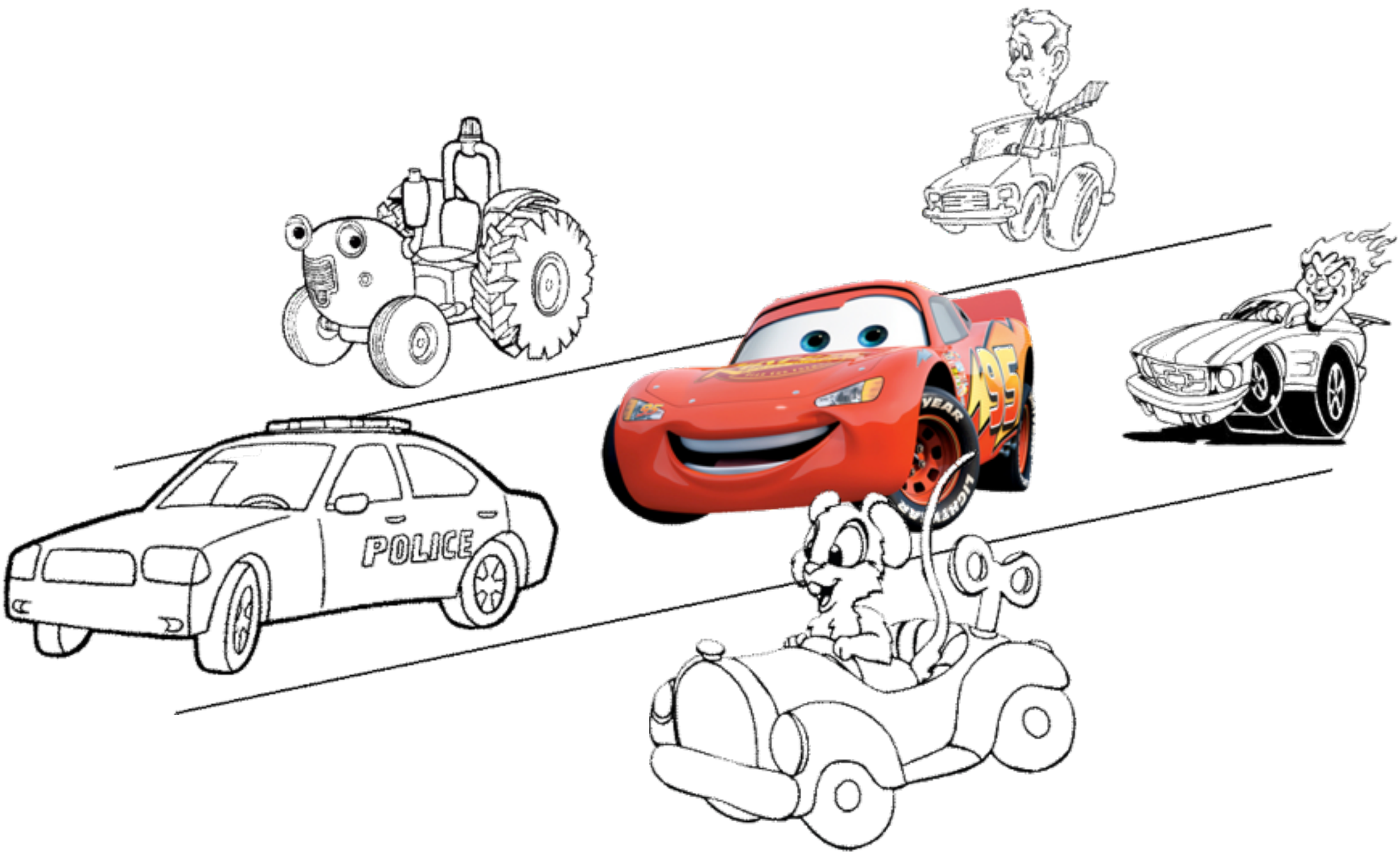
An Exploration of Group and Ring Signatures

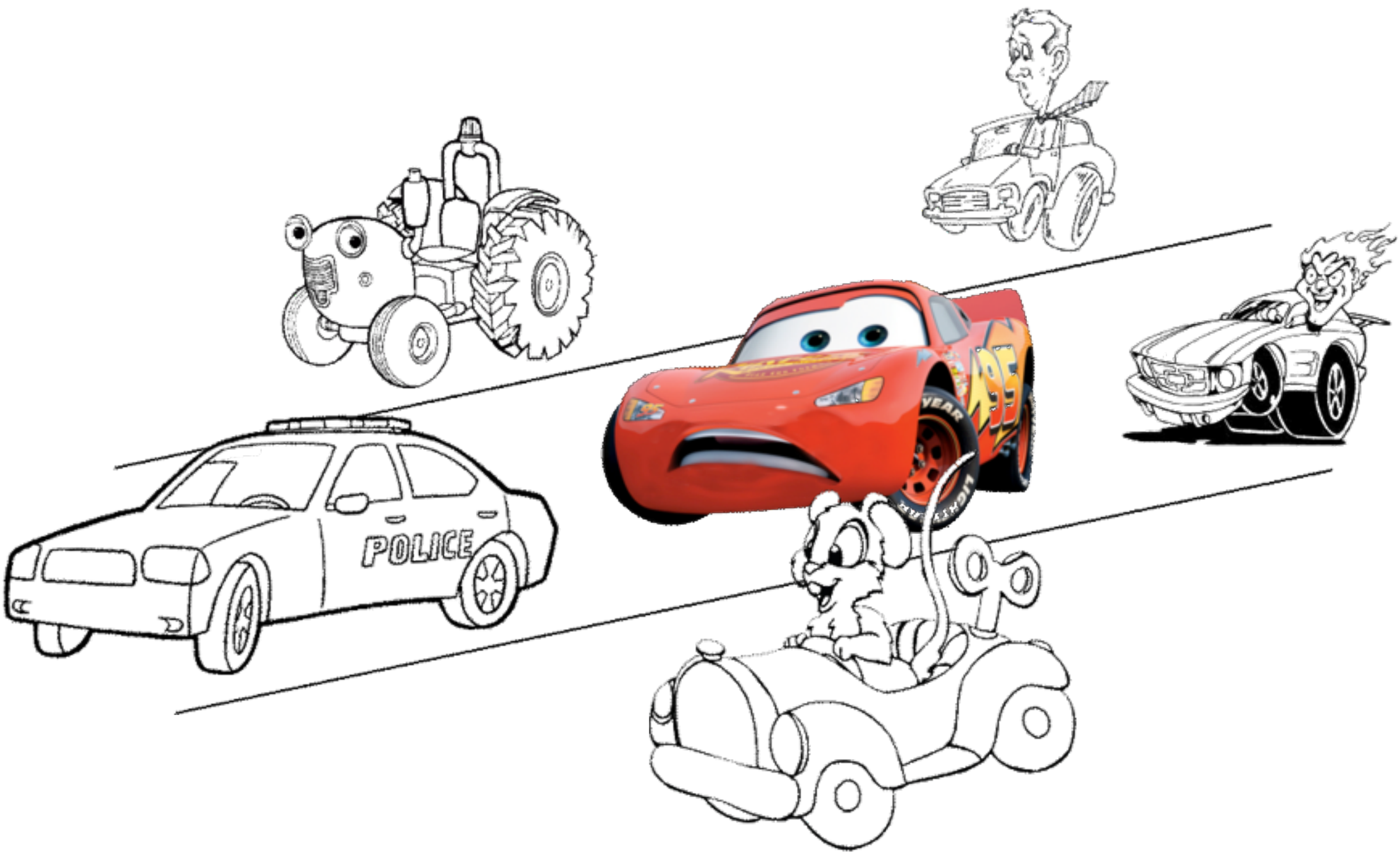
Sarah Meiklejohn

UC San Diego Research Exam
4 February 2011

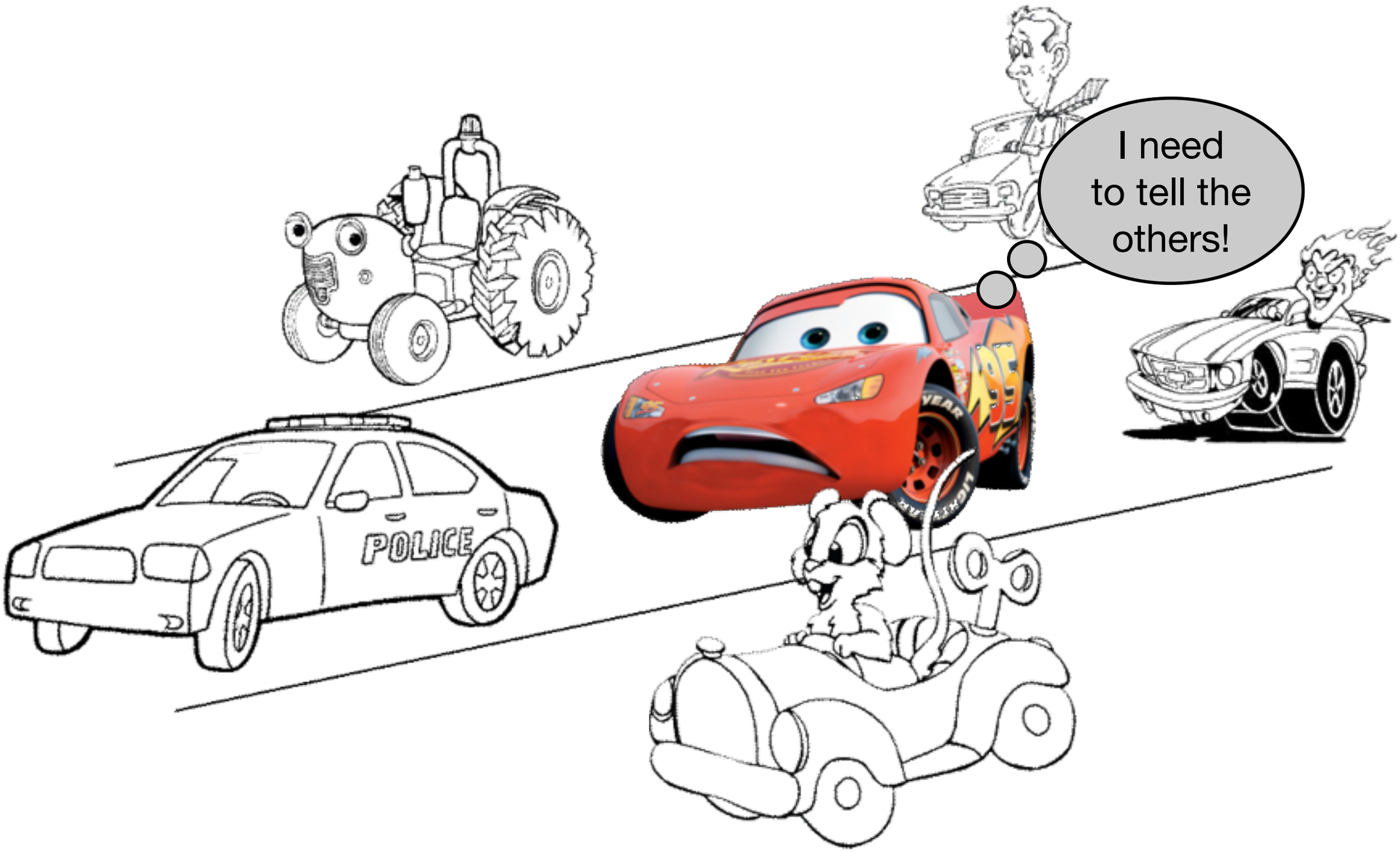
A real-world problem



A real-world problem

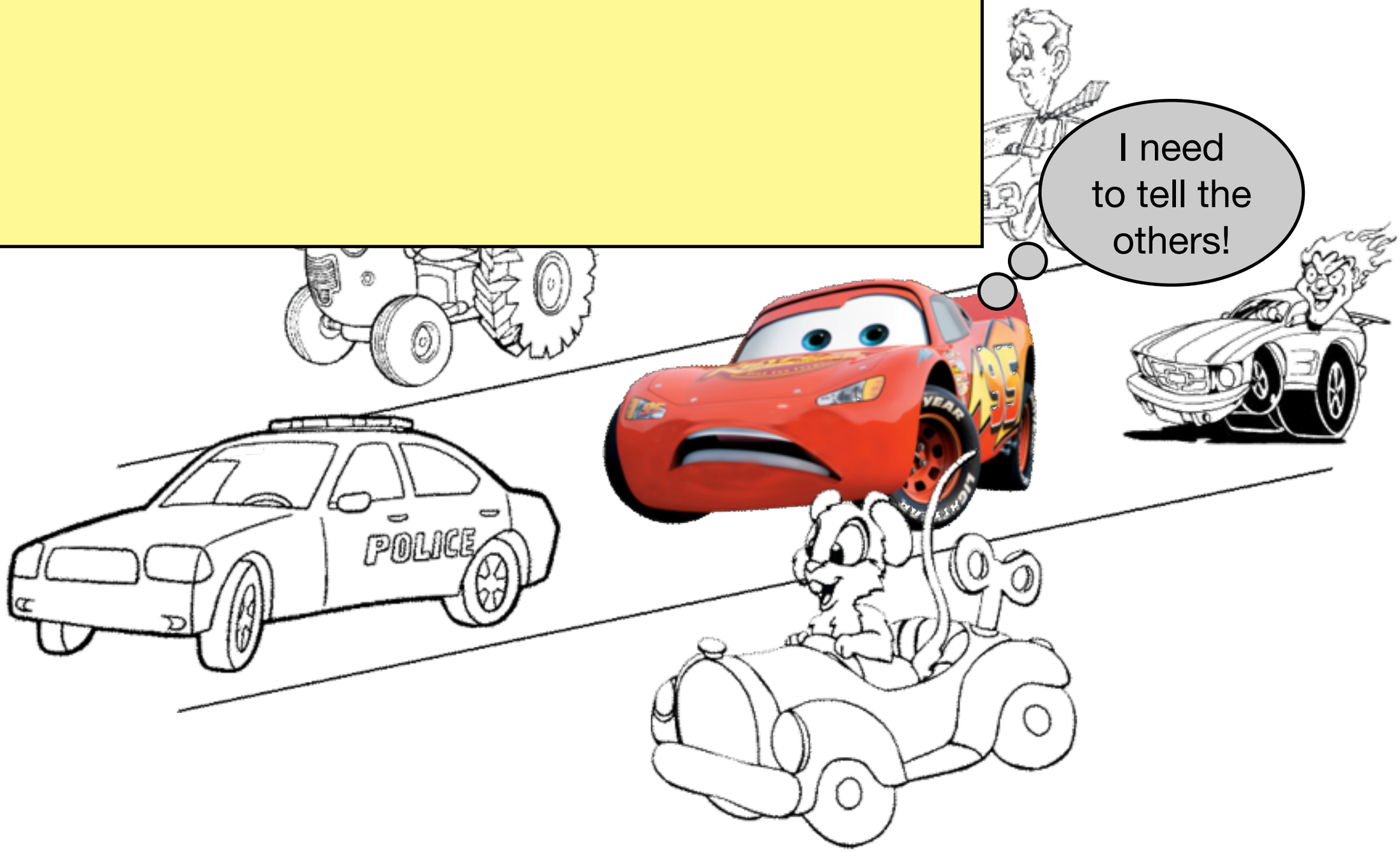


A real-world problem



A real-world problem

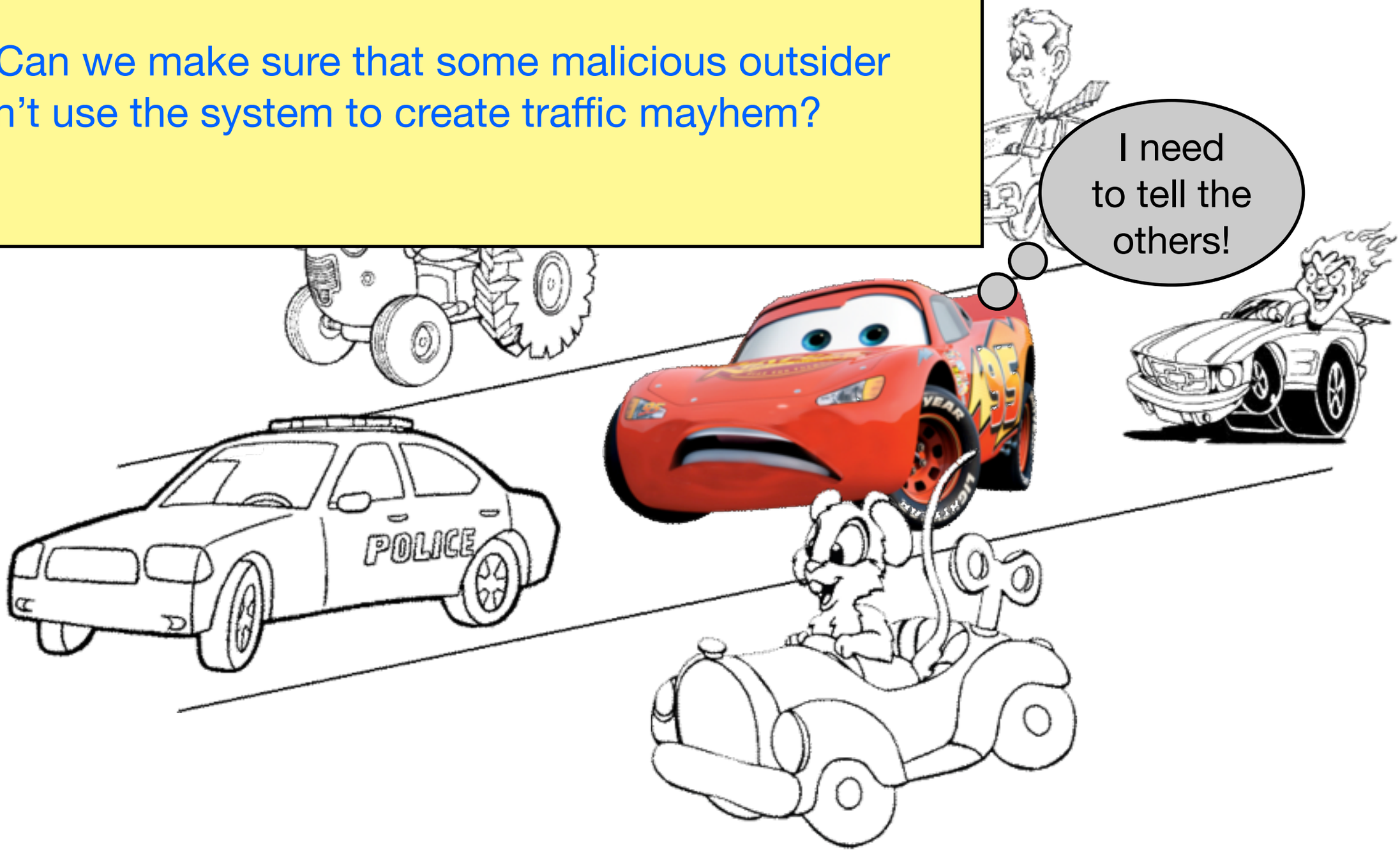
1. How can we communicate with the other cars?



A real-world problem

1. How can we communicate with the other cars?

2. Can we make sure that some malicious outsider can't use the system to create traffic mayhem?



Outline

Outline

Cryptographic background

Outline

Cryptographic background

Group signatures

Outline

Cryptographic background

Group signatures

Ring signatures

Outline

Cryptographic background

Group signatures

Ring signatures

Open problems

Outline

Cryptographic background

Group signatures

Ring signatures

Open problems

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer

Recipient

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer

Recipient

- Signer first runs an algorithm KeyGen to get signing keypair (pk,sk), ...

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer Recipient

- Signer first runs an algorithm KeyGen to get signing keypair (pk,sk), ...

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer Recipient

- Signer first runs an algorithm KeyGen to get signing keypair (pk,sk), ...

- ...then he can compute σ = Sign(sk,m) for the desired message m, and ...

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer Recipient

- Signer first runs an algorithm KeyGen to get signing keypair (pk,sk), ...

- ...then he can compute σ = Sign(sk,m) for the desired message m, and ...

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

- Signer first runs an algorithm KeyGen to get signing keypair (pk,sk), ...

- ...then he can compute σ = Sign(sk,m) for the desired message m, and ...

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer Recipient

- Signer first runs an algorithm KeyGen to get signing keypair (pk,sk), ...
- ...then he can compute σ = Sign(sk,m) for the desired message m, and ...
- Recipient can run Verify(pk,σ,m) to be sure σ was created by Signer

Digital signatures

- Signatures: Signer wants to send a message to Recipient, but wants to make sure she knows the message really came from him

Signer

Recipient

- We need signatures to be unforgeable, which means an adversary cannot successfully pretend to be the Signer (without knowing sk)

Outline

Cryptographic background

Group signatures

Intuition and motivation

Formal definitions

Extensions and variants

Comparison of existing schemes

Ring signatures

Open problems

Group signatures: why do we want them?

Group signatures: why do we want them?

Group 1

Group signatures: why do we want them?

pk_A, sk_A

pk_B, sk_B

pk_C, sk_C

pk_D, sk_D

Group 1

Group signatures: why do we want them?

pk_A, sk_A

pk_B, sk_B

pk_C, sk_C

pk_D, sk_D

Group 1

<u>Group 1</u>

Alice: pk_A

Bob: pk_B

Charlie: pk_C

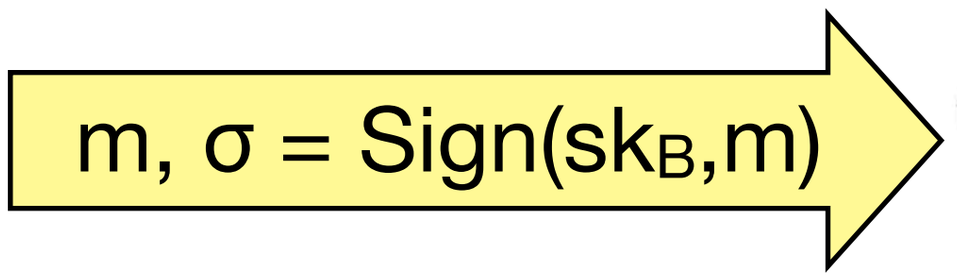
Dora: pk_D

Group signatures: why do we want them?

pk_A, sk_A pk_B, sk_B

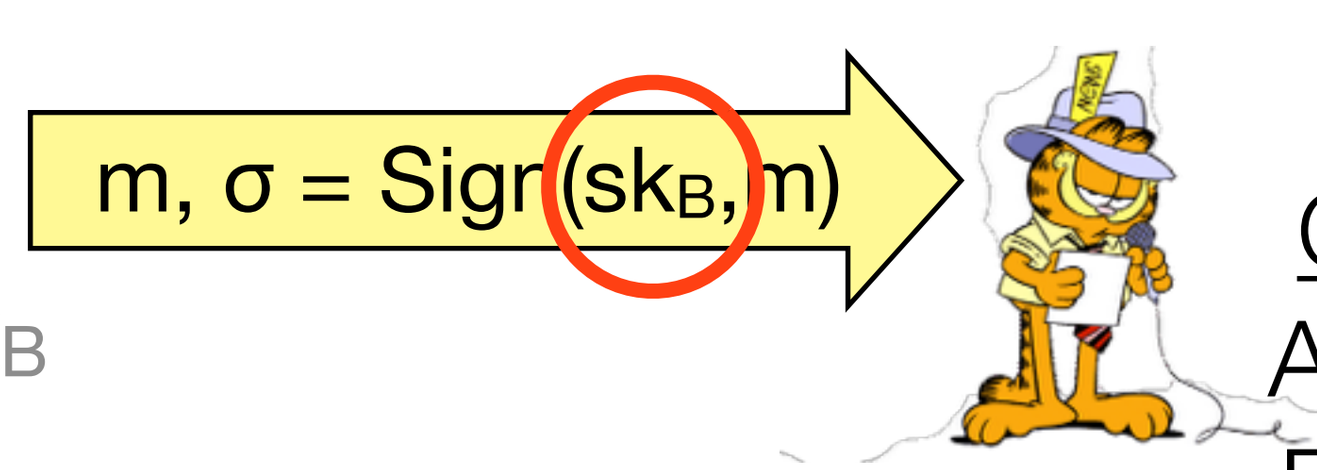
pk_C, sk_C pk_D, sk_D

Group 1

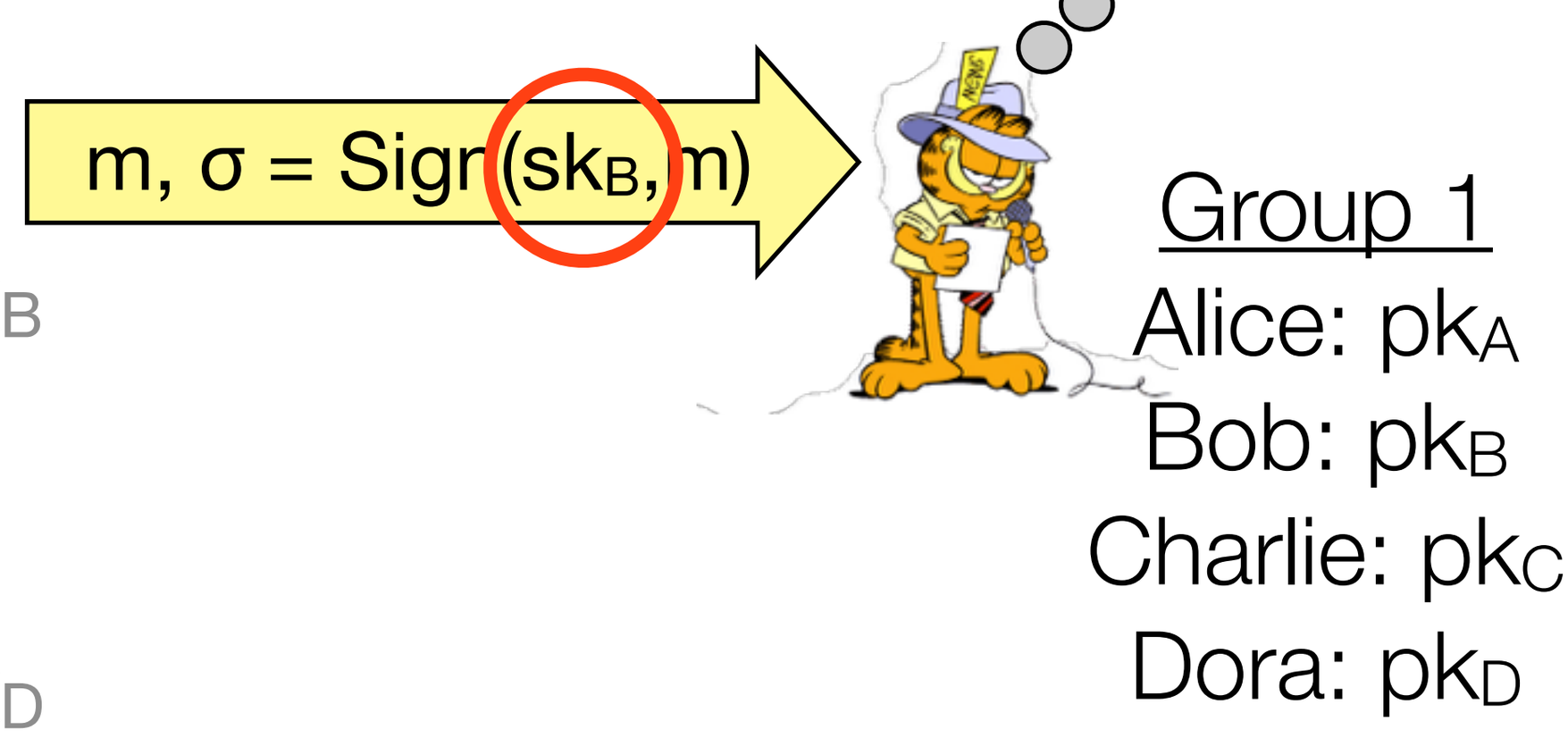


Group 1
Alice: pk_A
Bob: pk_B
Charlie: pk_C
Dora: pk_D

Group signatures: why do we want them?



Group signatures: why do we want them?



Group signatures: why do we want them?

pk_A, sk_A

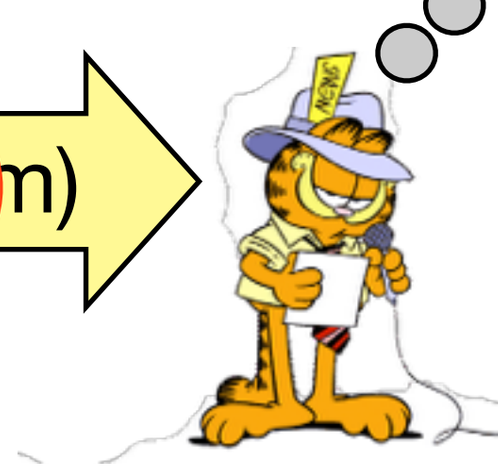
pk_B, sk_B

pk_C, sk_C

pk_D, sk_D

Group 1

$m, \sigma = Sign(sk_B, m)$



Verify(pk_B,σ,m) = 1... so Bob wrote the message!

<u>Group 1</u>

Alice: pk_A

Bob: pk_B

Charlie: pk_C

Dora: pk_D

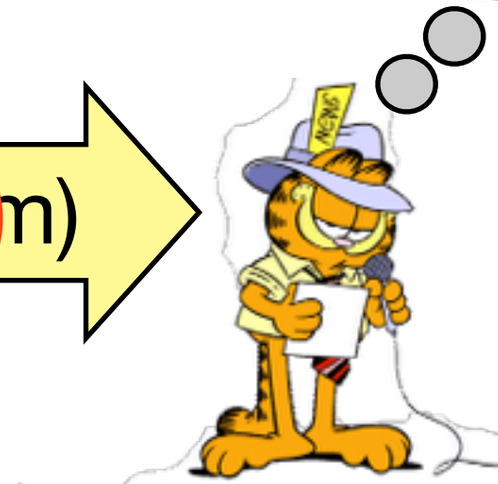
Group signatures: why do we want them?

pk_A, sk_A pk_B, sk_B

pk_C, sk_C pk_D, sk_D

Group 1

$m, \sigma = Sign(sk_B, m)$



Verify(pk_B,σ,m) = 1… so Bob wrote the message! **And he works for the CIA!**

<u>Group 1</u>

Alice: pk_A

Bob: pk_B

Charlie: pk_C

Dora: pk_D

Properties of group signatures: anonymity

Properties of group signatures: anonymity

Properties of group signatures: anonymity

sk_A

sk_B

sk_C

sk_D

Properties of group signatures: anonymity

pk_{CIA}

sk_A

sk_B

sk_C

sk_D

Properties of group signatures: anonymity

pk_{CIA}

sk_A

sk_B

sk_C

sk_D

Properties of group signatures: anonymity

pk_{CIA}

sk_A

sk_B

sk_C

sk_D

$m, \sigma = Sign(sk_B, m)$

Properties of group signatures: anonymity

pk_{CIA}

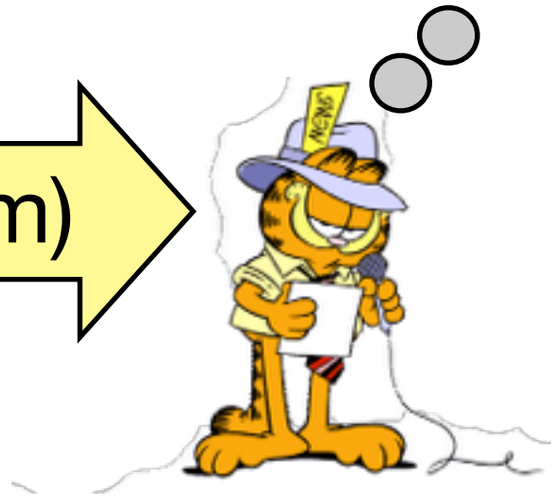
sk_A

sk_B

sk_C

sk_D

$m, \sigma = Sign(sk_B, m)$



Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA wrote the message.

Properties of group signatures: anonymity

pk_{CIA}

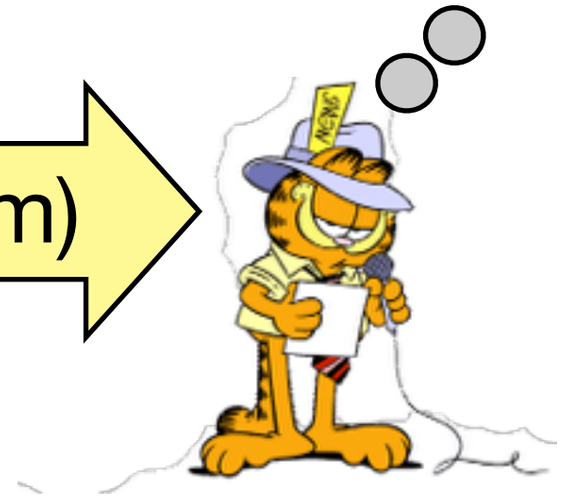
sk_A

sk_B

sk_C

sk_D

$m, \sigma = Sign(sk_B,m)$



Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA wrote the message.

- Given Sign(sk_B,m) and Sign(sk_D,m), recipient can't tell the difference
- This should be true even if he knows who has signed previous messages

Properties of group signatures: traceability

pk_{CIA}

sk_A

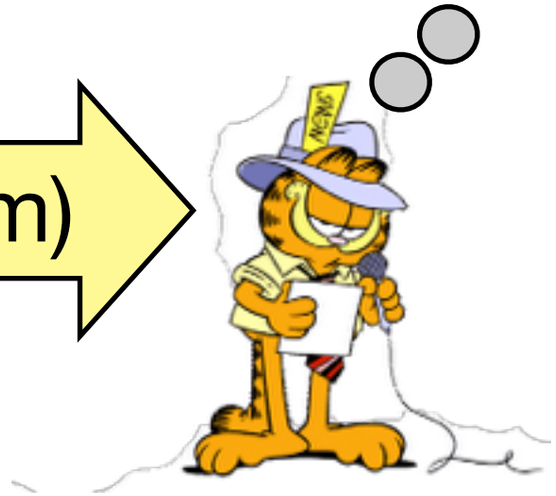
sk_B

sk_C

sk_D

$m, \sigma = Sign(sk_B, m)$

Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA
wrote the message.



Properties of group signatures: traceability

pk_{CIA}

sk_A

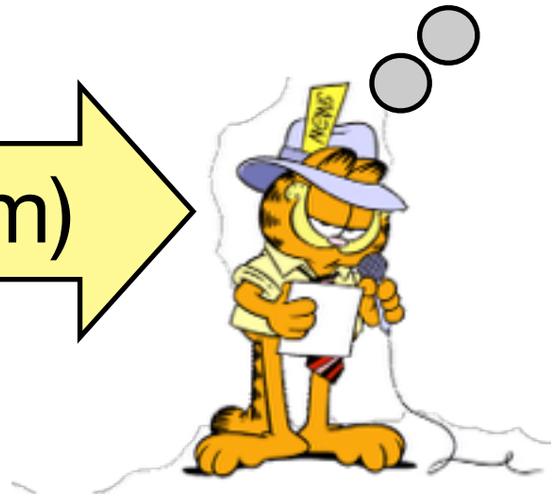
sk_B

sk_C

sk_D

$m, \sigma = Sign(sk_B, m)$

Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA
wrote the message.



Properties of group signatures: traceability

pk_{CIA}

sk_A

sk_B

sk_C

sk_D

m = “The CIA is the worst!”

$m, \sigma = Sign(sk_B, m)$

Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA wrote the message.

Properties of group signatures: traceability

pk_{CIA}

tk

sk_A

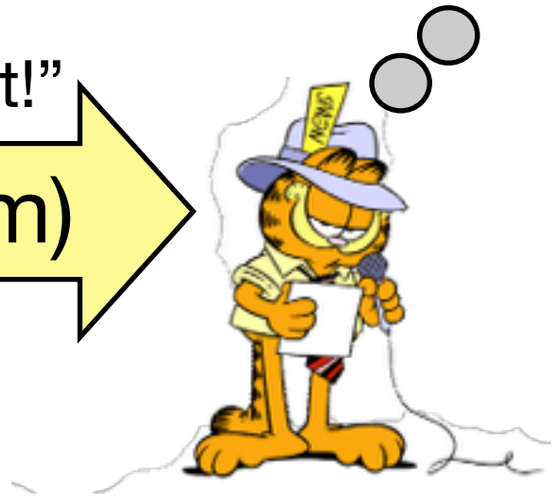
sk_B

sk_C

sk_D

m = “The CIA is the worst!”

$m, \sigma = Sign(sk_B, m)$



Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA wrote the message.

Properties of group signatures: traceability

pk_{CIA}

tk

Verify(pk_{CIA},σ,m) = 1...
so someone from the CIA wrote the message.

sk_A

sk_B

m = “The CIA is the worst!”

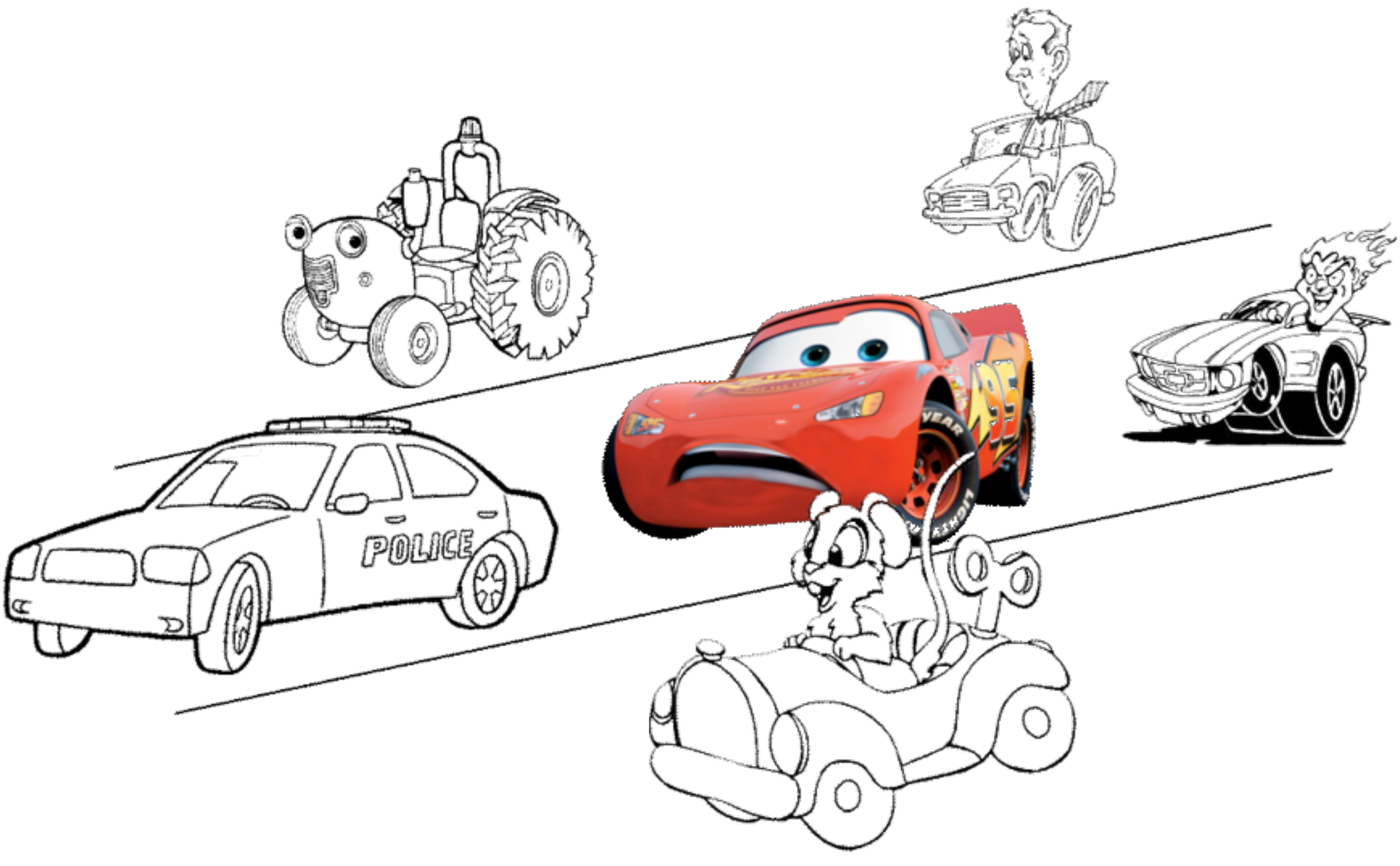
m, σ = Sign(sk_B,m)

sk_C

sk_D

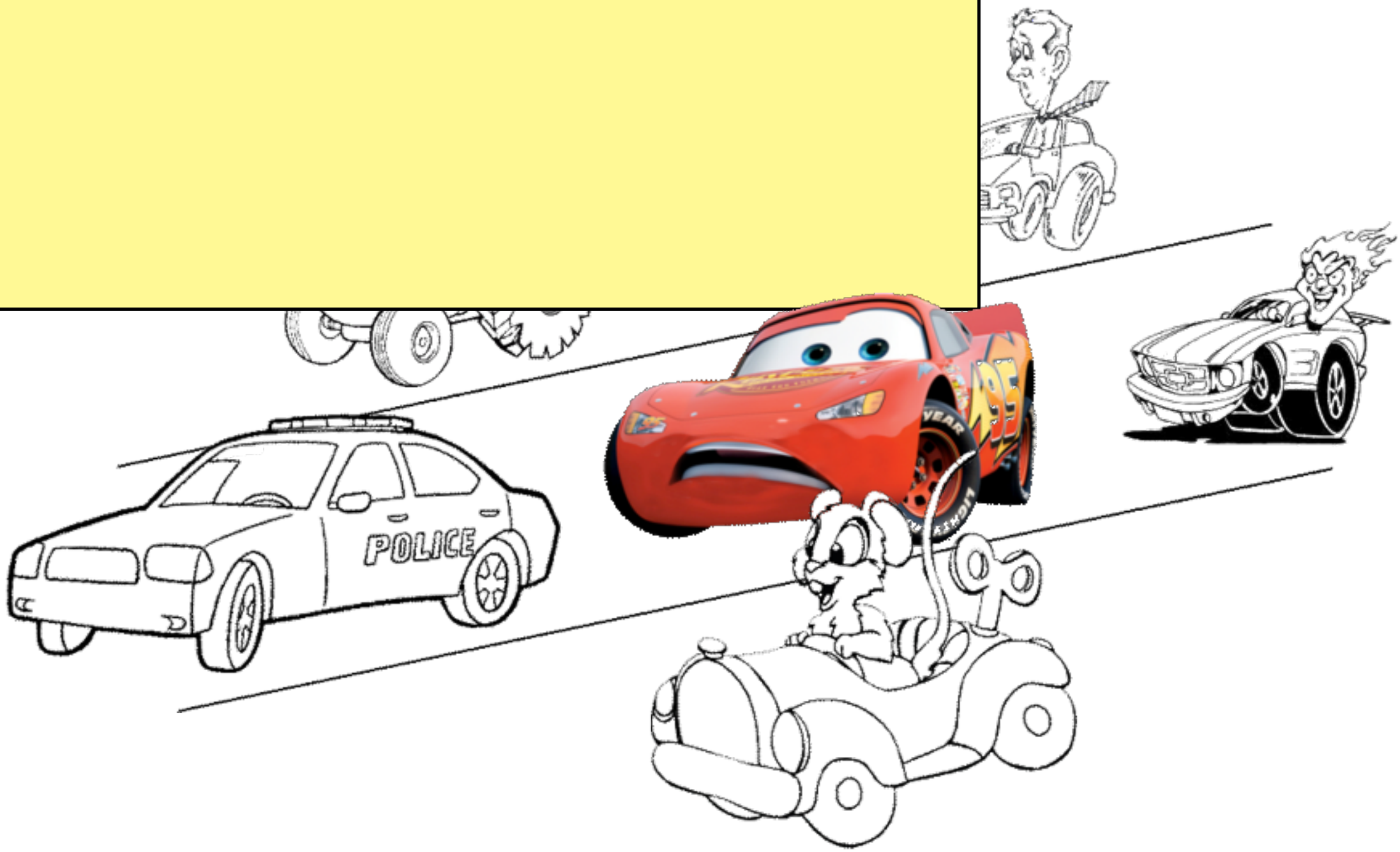
- Want new algorithm Trace s.t. Trace(tk,σ) = Bob
- Whoever has access to tk breaks anonymity

Using group signatures with our real-world problem



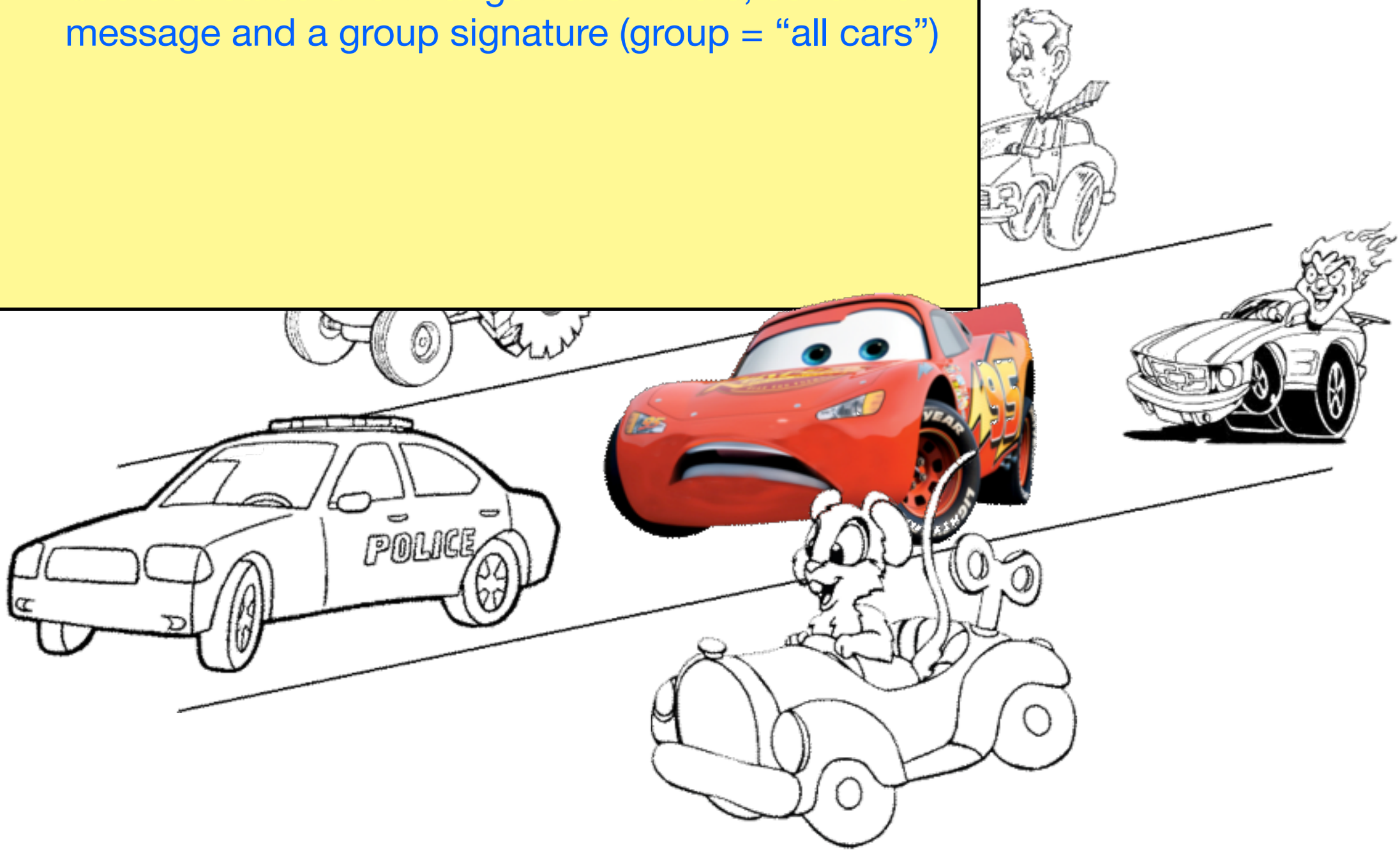
Using group signatures with our real-world problem

1. How can we communicate with the other cars?



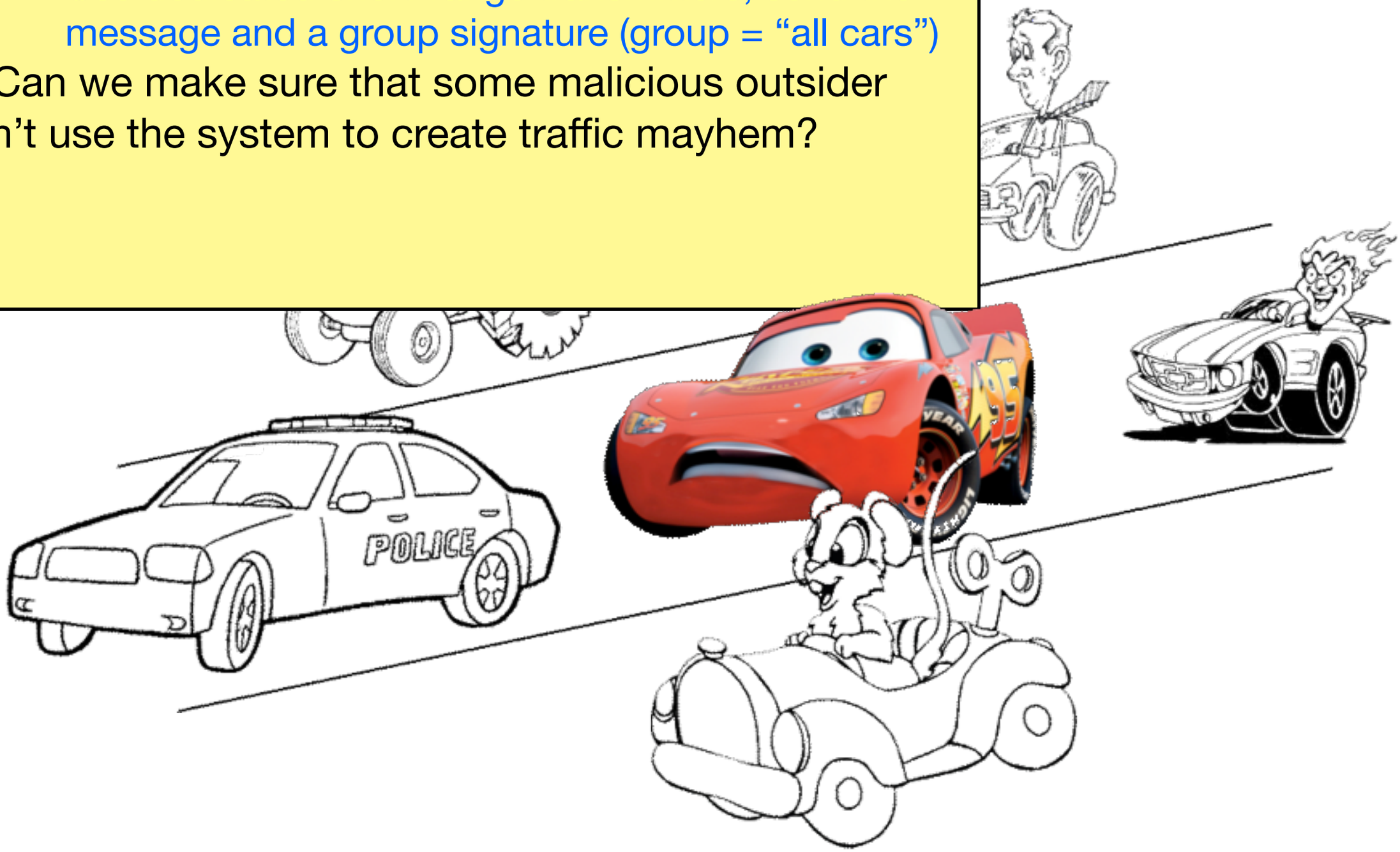
Using group signatures with our real-world problem

1. How can we communicate with the other cars?
   - Use dedicated short-range transmitters, send the message and a group signature (group = “all cars”)



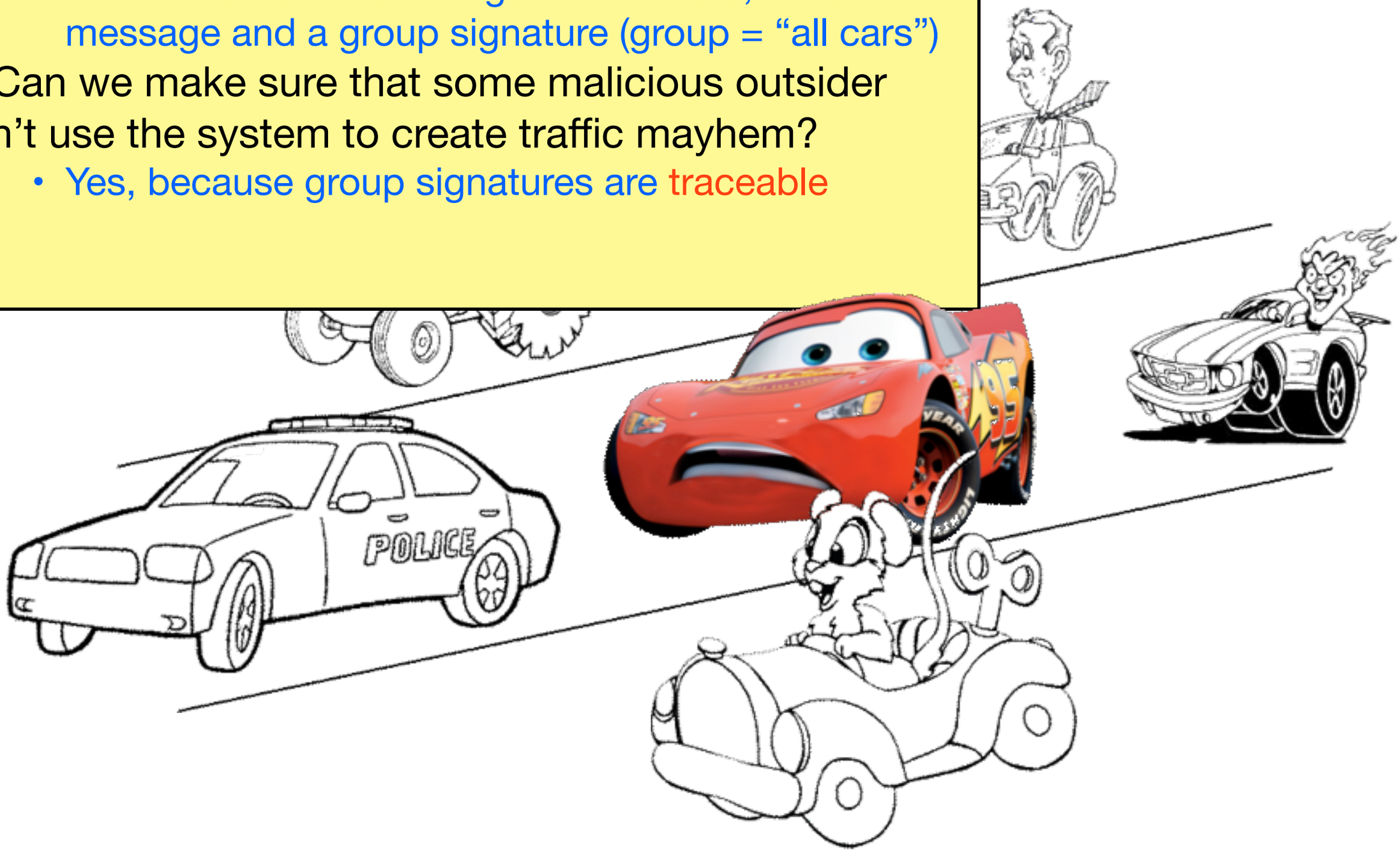
Using group signatures with our real-world problem

1. How can we communicate with the other cars?
    - Use dedicated short-range transmitters, send the message and a group signature (group = “all cars”)
2. Can we make sure that some malicious outsider can’t use the system to create traffic mayhem?



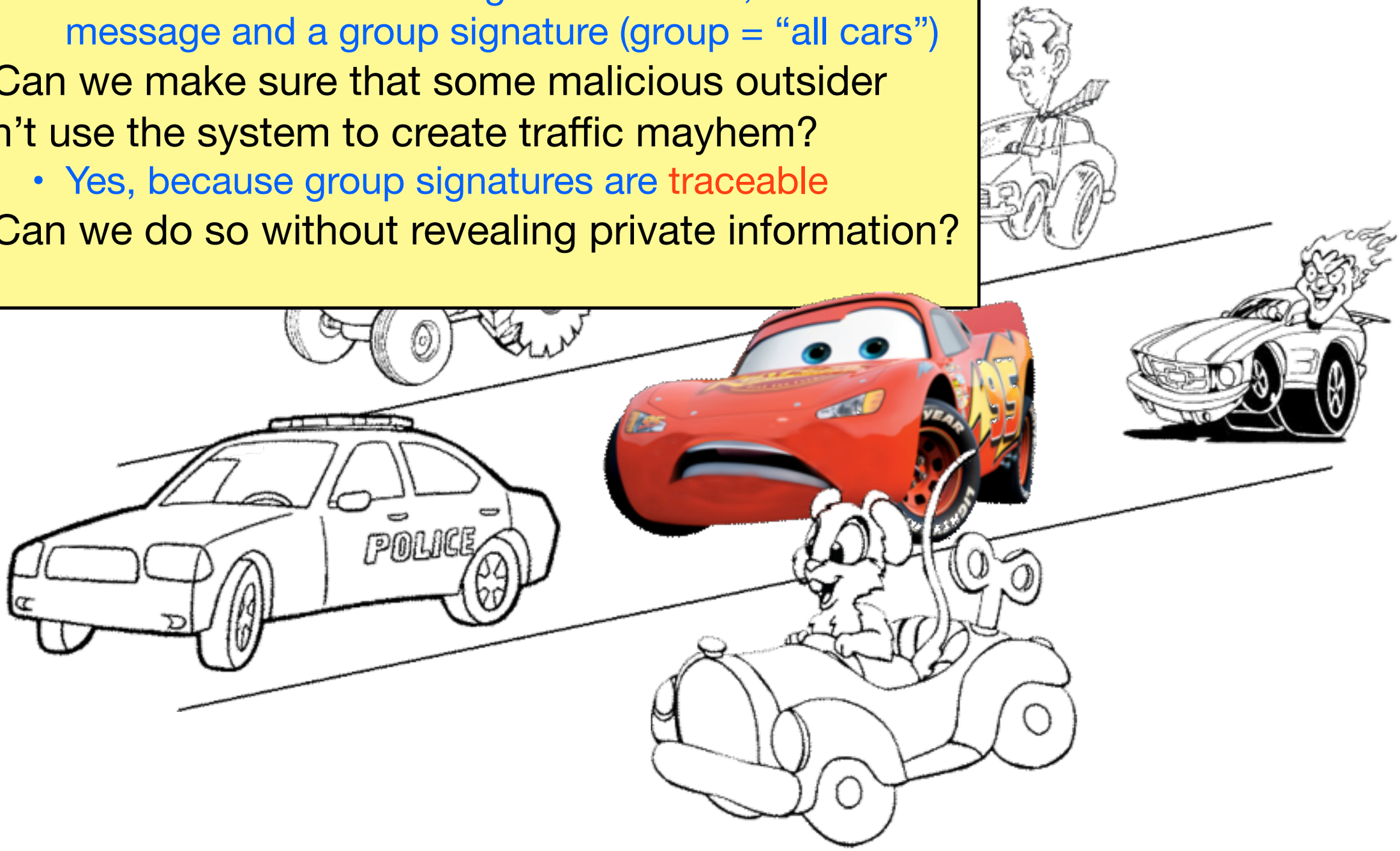
Using group signatures with our real-world problem

1. How can we communicate with the other cars?
    - Use dedicated short-range transmitters, send the message and a group signature (group = “all cars”)
2. Can we make sure that some malicious outsider can’t use the system to create traffic mayhem?
    - Yes, because group signatures are traceable



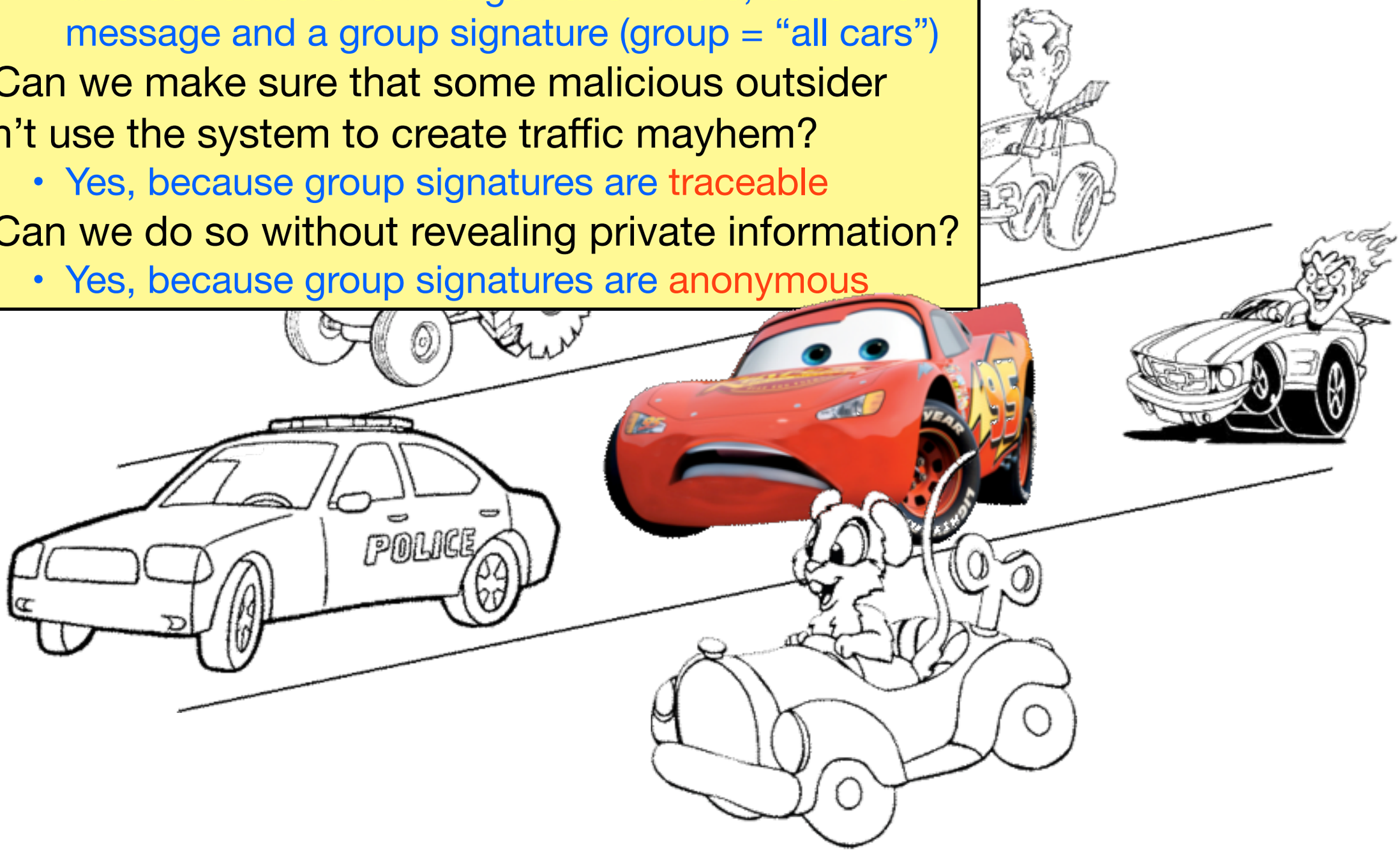
Using group signatures with our real-world problem

1. How can we communicate with the other cars?
    - Use dedicated short-range transmitters, send the message and a group signature (group = “all cars”)
2. Can we make sure that some malicious outsider can’t use the system to create traffic mayhem?
    - Yes, because group signatures are traceable
3. Can we do so without revealing private information?



Using group signatures with our real-world problem

1. How can we communicate with the other cars?
    - Use dedicated short-range transmitters, send the message and a group signature (group = “all cars”)
2. Can we make sure that some malicious outsider can’t use the system to create traffic mayhem?
    - Yes, because group signatures are traceable
3. Can we do so without revealing private information?
    - Yes, because group signatures are anonymous



Group signatures: a formal characterization

- A group signature is a tuple of algorithms (KeyGen,Sign,Verify,Trace)

Group signatures: a formal characterization

- A group signature is a tuple of algorithms (KeyGen,Sign,Verify,Trace)
- KeyGen(1^k,1^n): outputs group public key pk, master secret key msk, and signing keys for each user in the group $\{sk_i\}_i$

Group signatures: a formal characterization

- A group signature is a tuple of algorithms (KeyGen,Sign,Verify,Trace)
- KeyGen(1^k,1^n): outputs group public key pk, master secret key msk, and signing keys for each user in the group $\{sk_i\}_i$

- Sign(sk_i,m): outputs signature σ on message m

Group signatures: a formal characterization

- A group signature is a tuple of algorithms (KeyGen,Sign,Verify,Trace)
- KeyGen(1^k,1^n): outputs group public key pk, master secret key msk, and signing keys for each user in the group $\{sk_i\}_i$

- Sign(sk_i,m): outputs signature σ on message m
- Verify(pk,σ,m): checks that σ is a valid signature on m formed by some member of the group defined by pk (and outputs 1 if yes and 0 if no)

Group signatures: a formal characterization

- A group signature is a tuple of algorithms (KeyGen,Sign,Verify,Trace)
- KeyGen(1^k,1^n): outputs group public key pk, master secret key msk, and signing keys for each user in the group $\{sk_i\}_i$

- Sign(sk_i,m): outputs signature σ on message m

- Verify(pk,σ,m): checks that σ is a valid signature on m formed by some member of the group defined by pk (and outputs 1 if yes and 0 if no)

- Trace(msk,σ,m): outputs either index i such that σ = Sign(sk_i,m) or ⊥ to indicate failure (or that Verify(pk,σ,m) = 0)

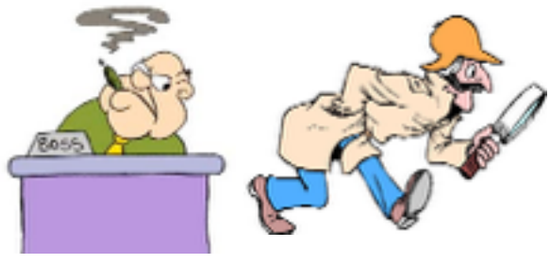
Group signatures: a formal characterization

- A group signature is a tuple of algorithms (KeyGen,Sign,Verify,Trace)
- KeyGen(1^k,1^n): outputs group public key pk, master secret key msk, and signing keys for each user in the group $\{sk_i\}_i$

- Sign(sk_i,m): outputs signature σ on message m

- Verify(pk,σ,m): checks that σ is a valid signature on m formed by some member of the group defined by pk (and outputs 1 if yes and 0 if no)

- Trace(msk,σ,m): outputs either index i such that σ = Sign(sk_i,m) or ⊥ to indicate failure (or that Verify(pk,σ,m) = 0)



Anonymity: a more formal definition

Game **G**

Anonymity: a more formal definition

Game **G**

Adversary **A**

Anonymity: a more formal definition

Game **G**

Adversary **A**

Anonymity: a more formal definition

Game **G**

Adversary **A**

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

$pk,msk,\{sk_i\} \leftarrow KeyGen(1^k,1^n)$

Adversary **A**

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

$pk,msk,\{sk_i\} \leftarrow KeyGen(1^k,1^n)$

Adversary **A**

$pk, \{sk_i\}$

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

msk

Adversary **A**

pk, {sk_i}

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

msk

Sign(sk_B,m)

Adversary **A**

pk, {sk_i}

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

msk

B

Sign(sk_B,m)

Adversary **A**

pk, {sk_i}

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

msk

Sign(sk_i,m)

Adversary **A**

pk, {sk_i}

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

Game **G**

msk

i

Sign(sk_i,m)

Adversary **A**

pk, {sk_i}

Phase 1: getting to see who signed which messages

Anonymity: a more formal definition

pk, $\{sk_i\}$

Phase 2: picking identities and receiving a challenge

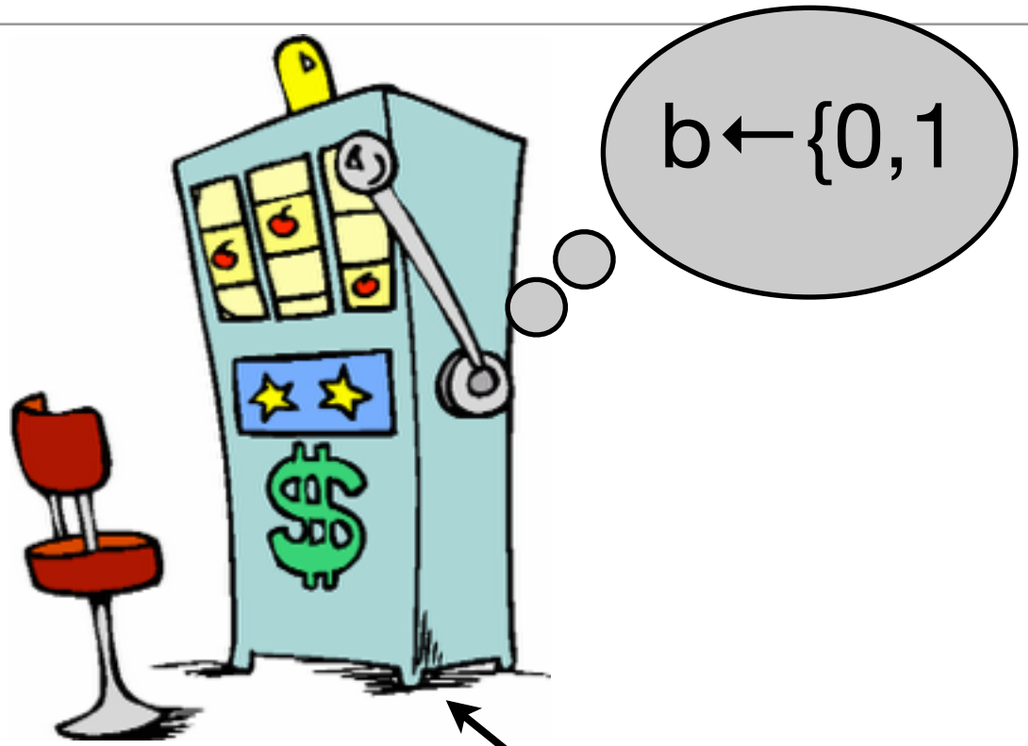
Anonymity: a more formal definition

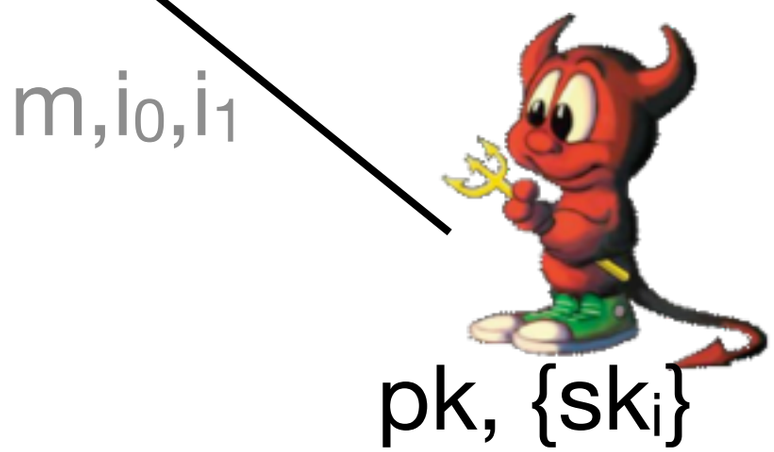
m, i_0, i_1

$pk, \{sk_i\}$

Phase 2: picking identities and receiving a challenge

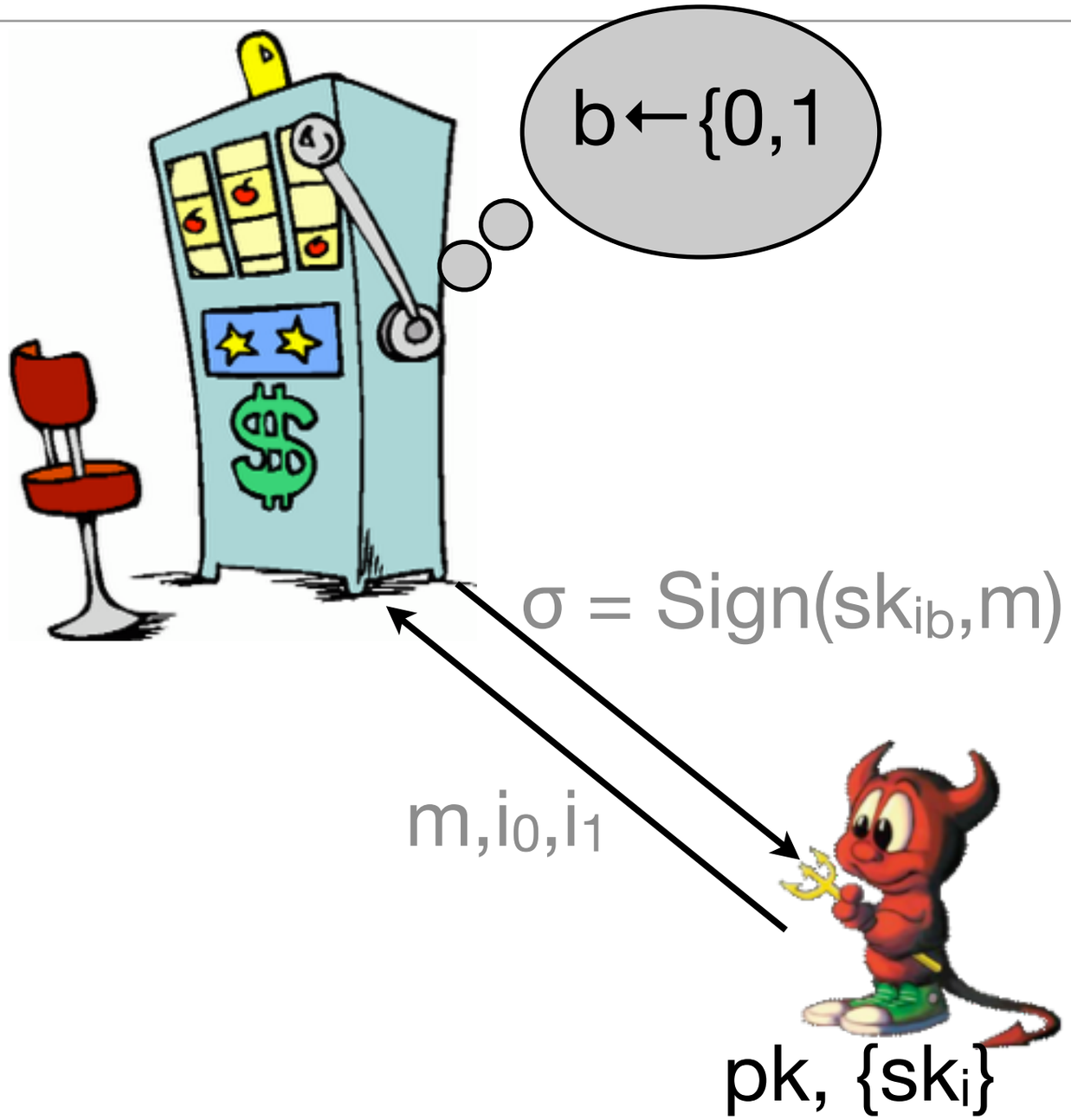
Anonymity: a more formal definition





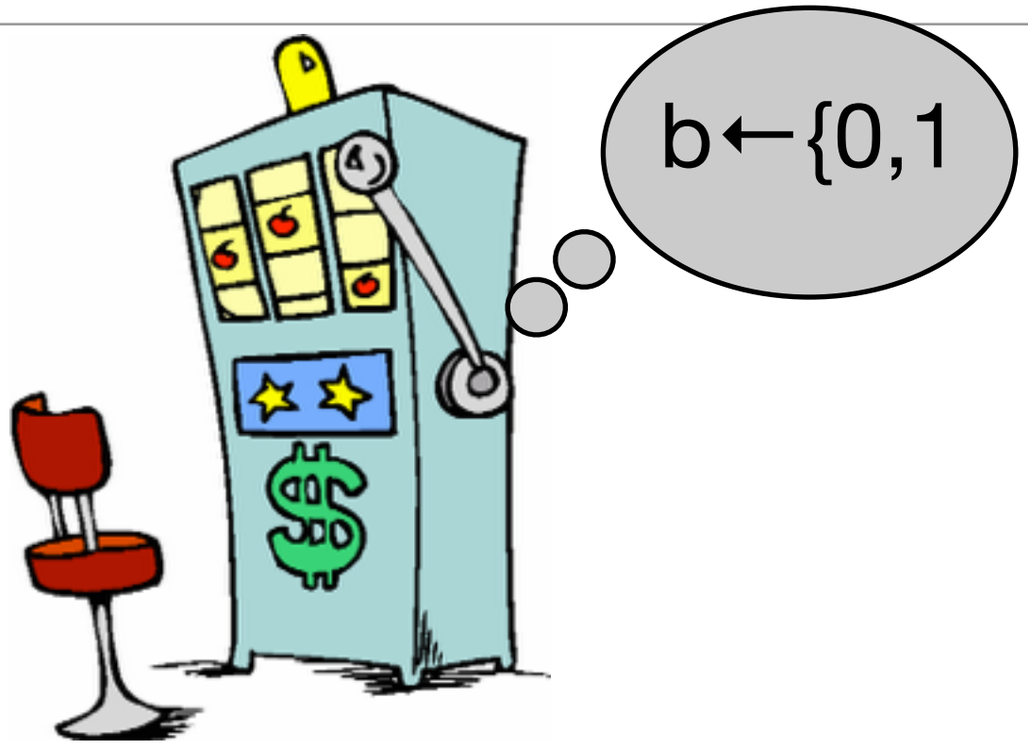
Phase 2: picking identities and receiving a challenge

Anonymity: a more formal definition



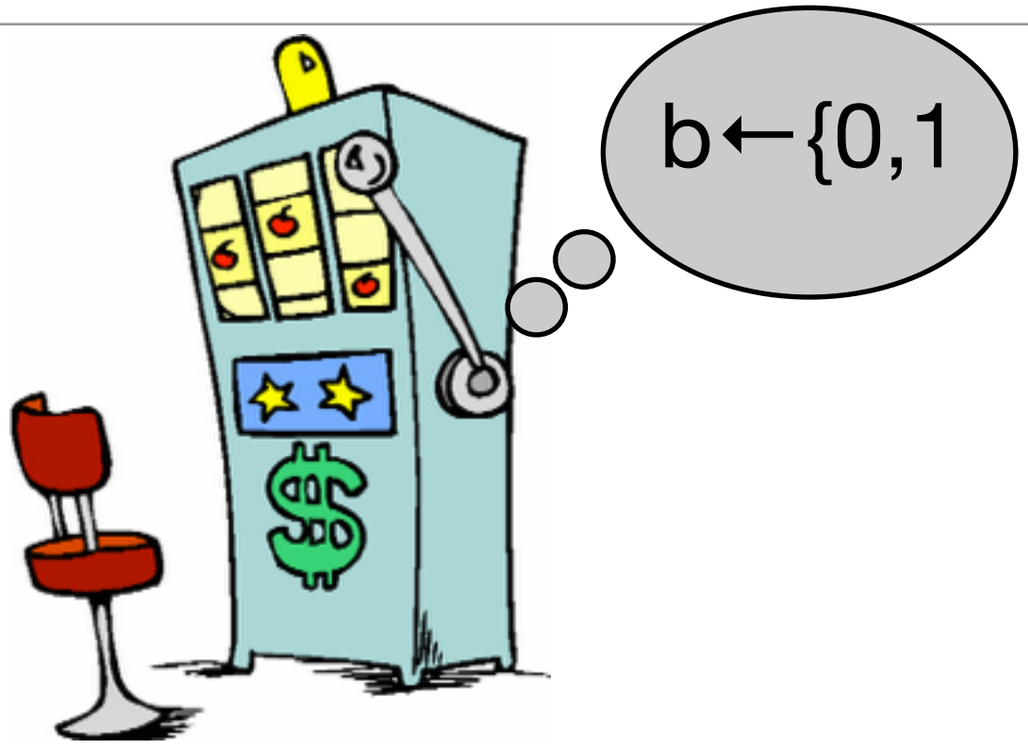
Phase 2: picking identities and receiving a challenge

Anonymity: a more formal definition



$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

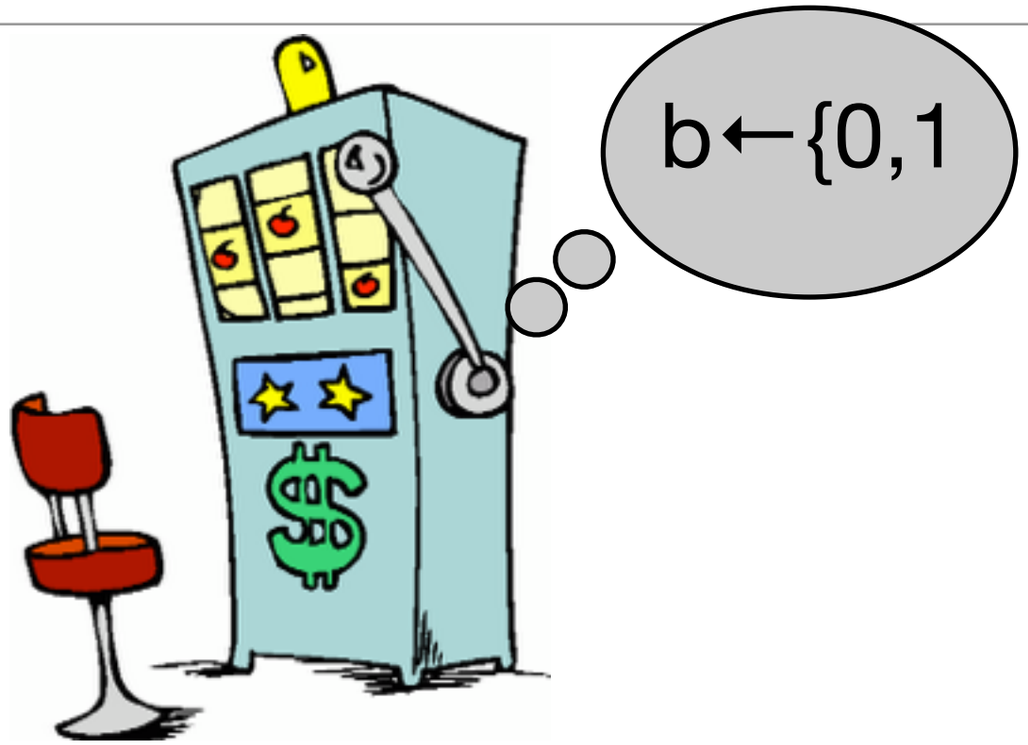
Anonymity: a more formal definition



$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

Phase 3: getting to see who signed which messages (again)

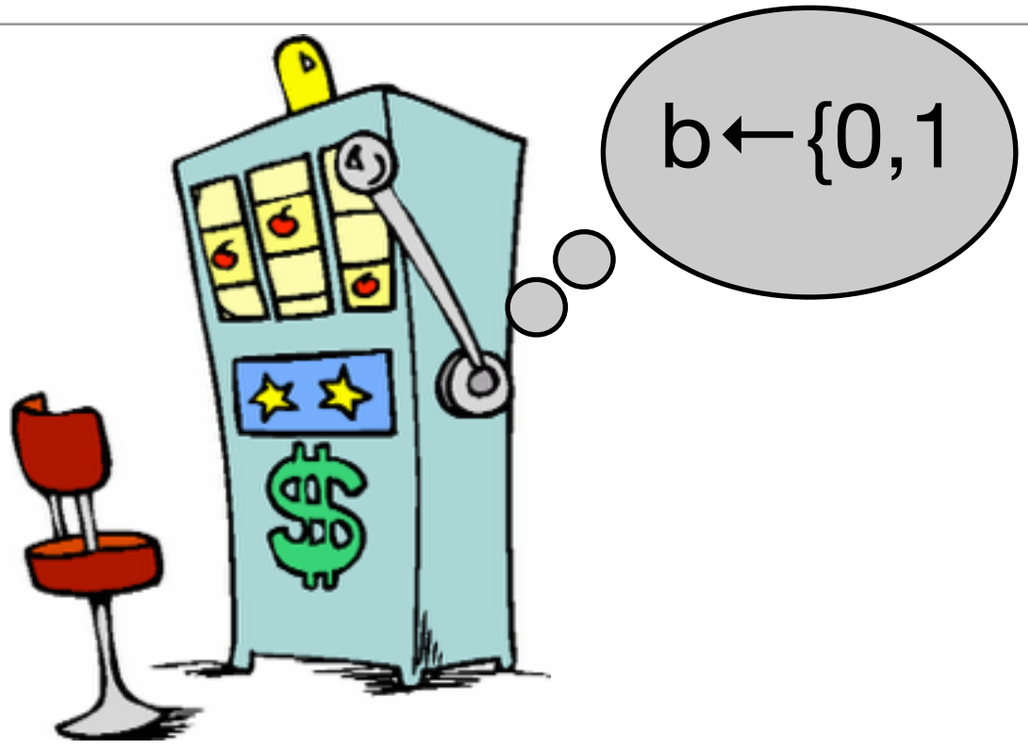
Anonymity: a more formal definition



$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

Phase 3: getting to see who signed which messages (again)

Anonymity: a more formal definition

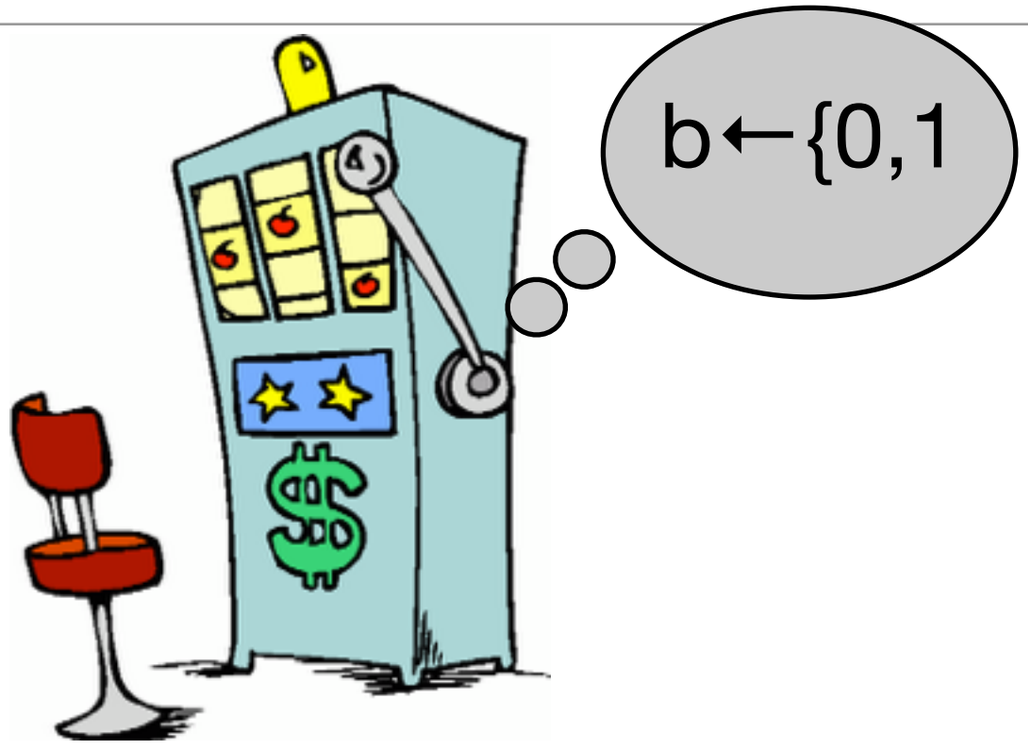


Sign(sk_i,m) ≠ σ

pk, {sk_i}, σ = Sign(sk_{ib},m)

Phase 3: getting to see who signed which messages (again)

Anonymity: a more formal definition



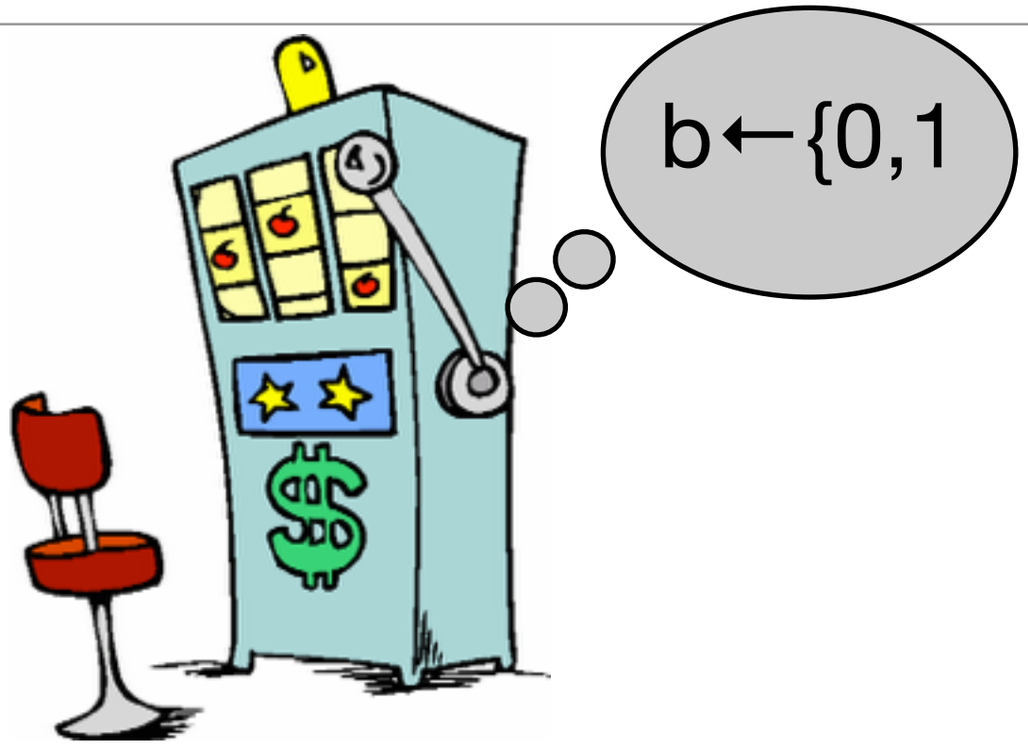
i

$Sign(sk_i,m) \neq \sigma$

$pk, \{sk_i\}, \sigma = Sign(sk_{ib},m)$

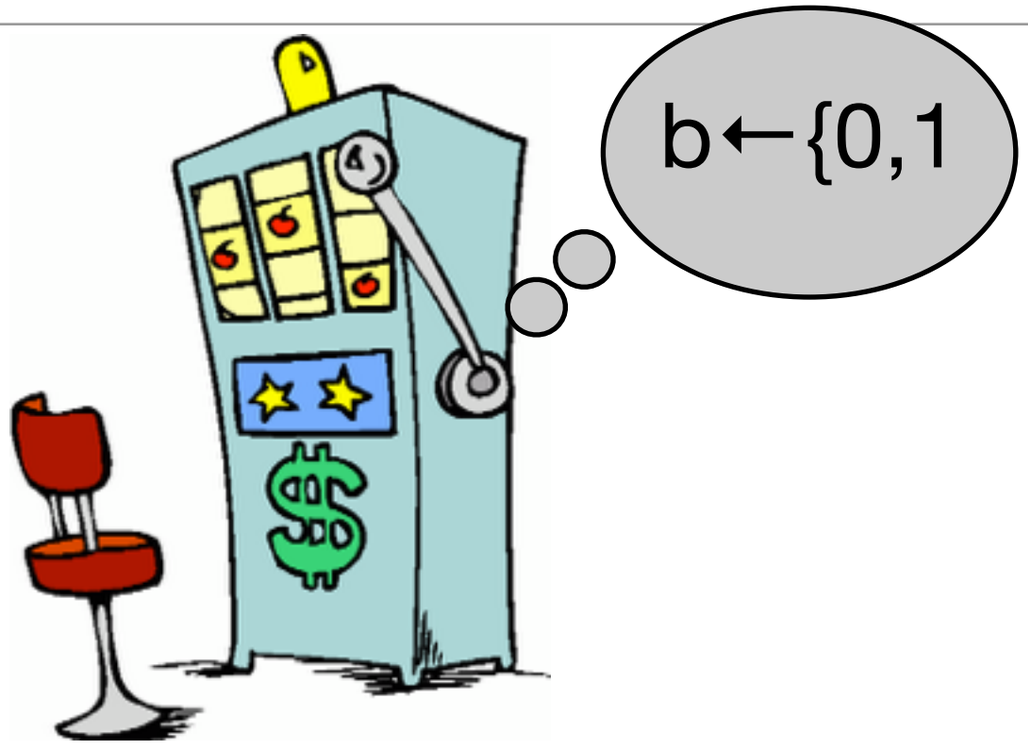
Phase 3: getting to see who signed which messages (again)

Anonymity: a more formal definition



$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

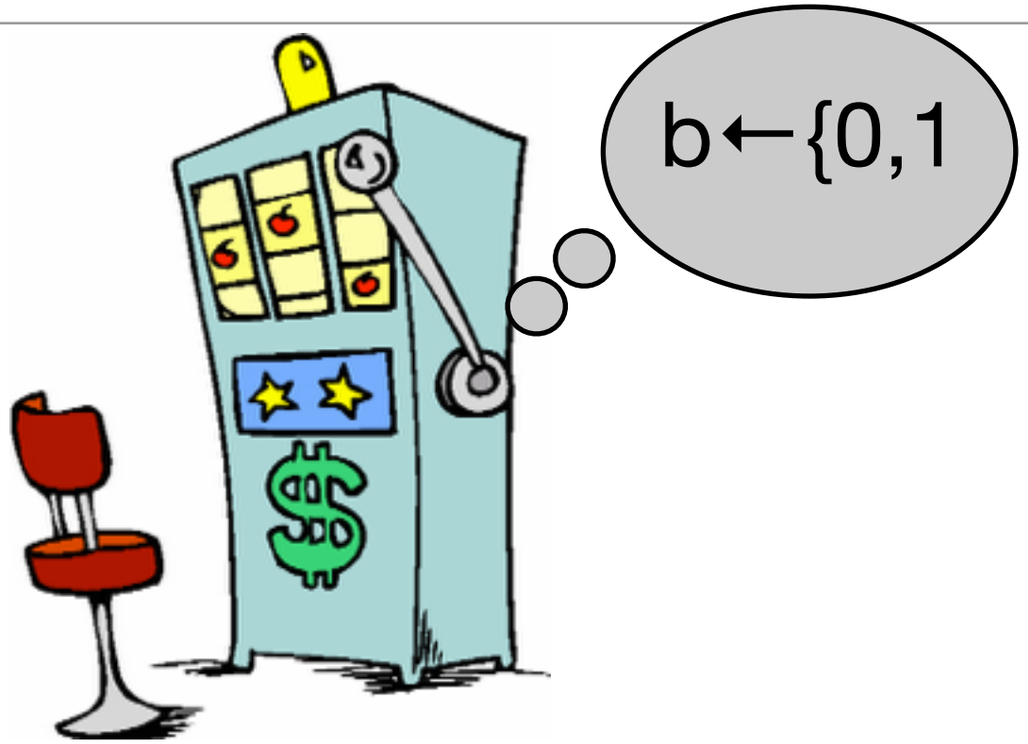
Anonymity: a more formal definition

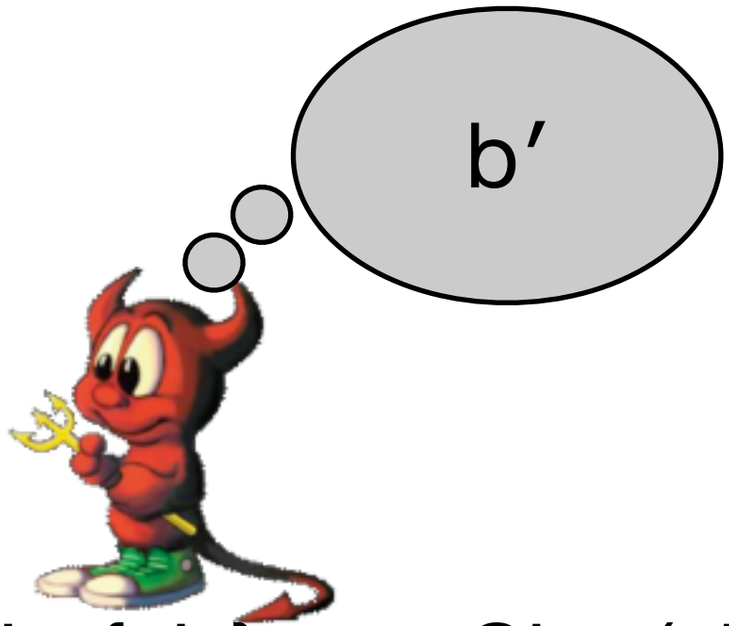


$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

Phase 4: guessing the bit b

Anonymity: a more formal definition

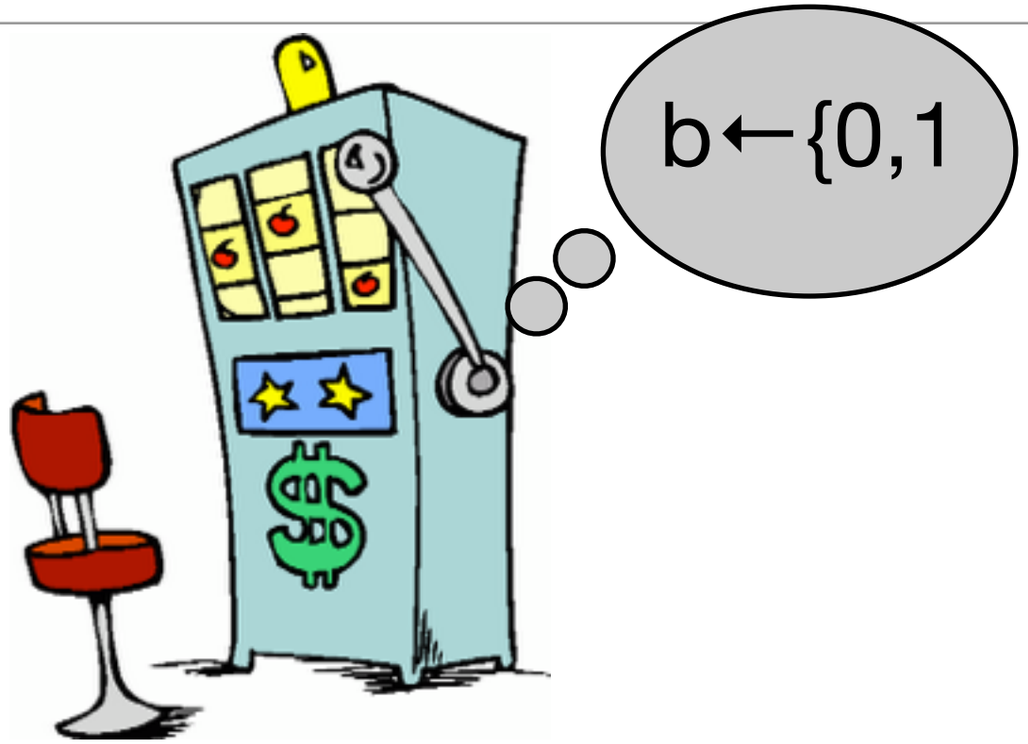




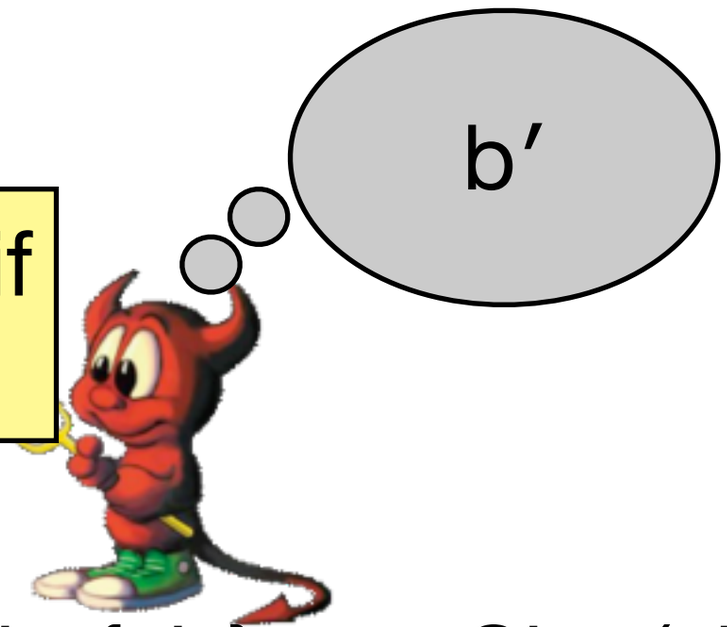
$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

Phase 4: guessing the bit b

Anonymity: a more formal definition



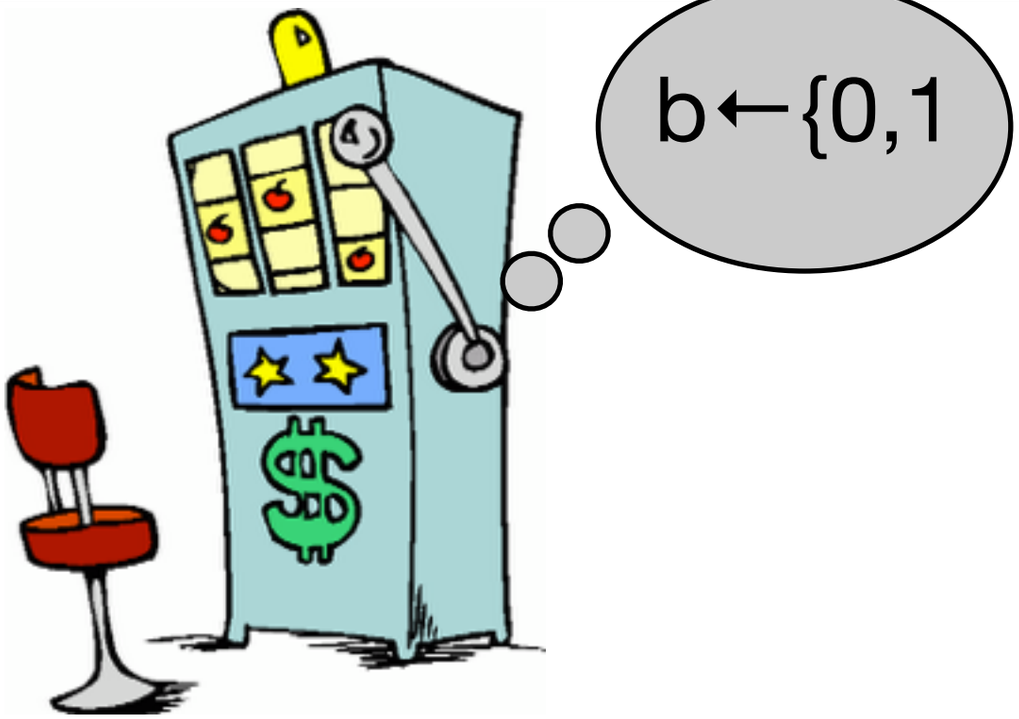
We say that A wins at G if
b = b′



$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

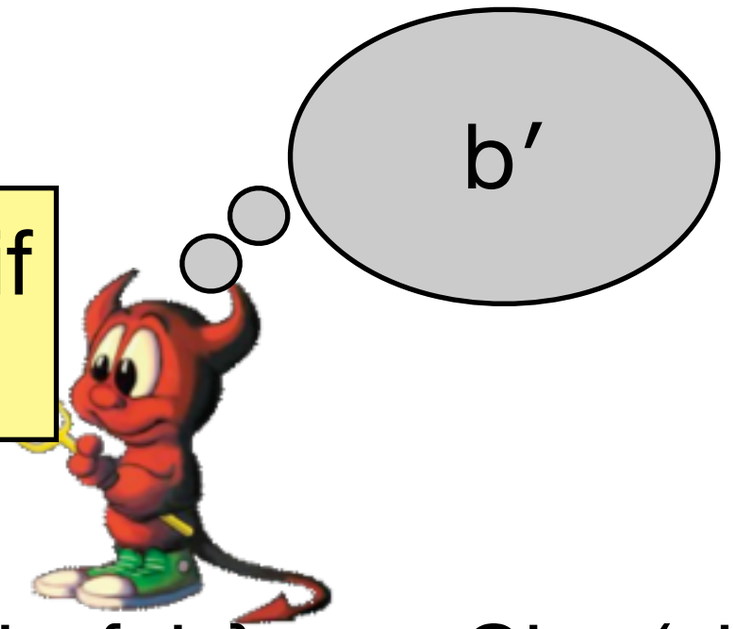
Phase 4: guessing the bit b

Anonymity: a more formal definition



Say that scheme is anonymous if the probability that A wins at G is very small (negligible)

We say that A wins at G if b = b′



$pk, \{sk_i\}, \sigma = Sign(sk_{ib}, m)$

Phase 4: guessing the bit b

Traceability: a more formal definition

Traceability: a more formal definition

Traceability: a more formal definition

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

C

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

$pk,msk,\{sk_i\} \leftarrow KeyGen(1^k,1^n)$

C

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

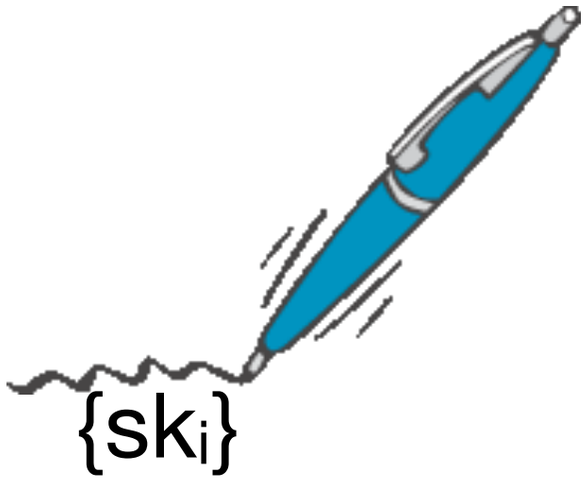
$pk,msk,\{sk_i\} \leftarrow KeyGen(1^k,1^n)$

pk, msk

C

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

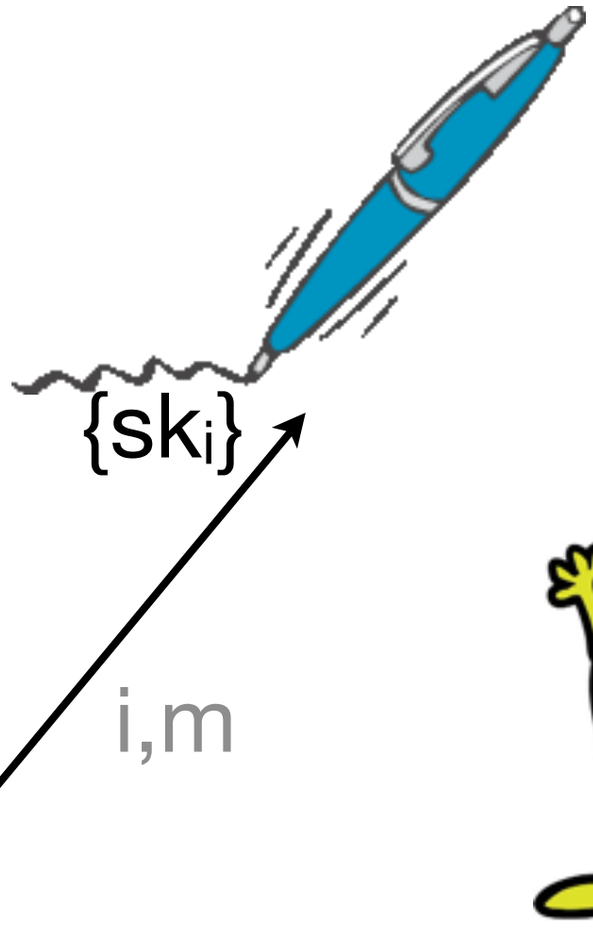


pk, msk

C

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition



pk, msk

C

Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition

Sign(sk$_i$,m)

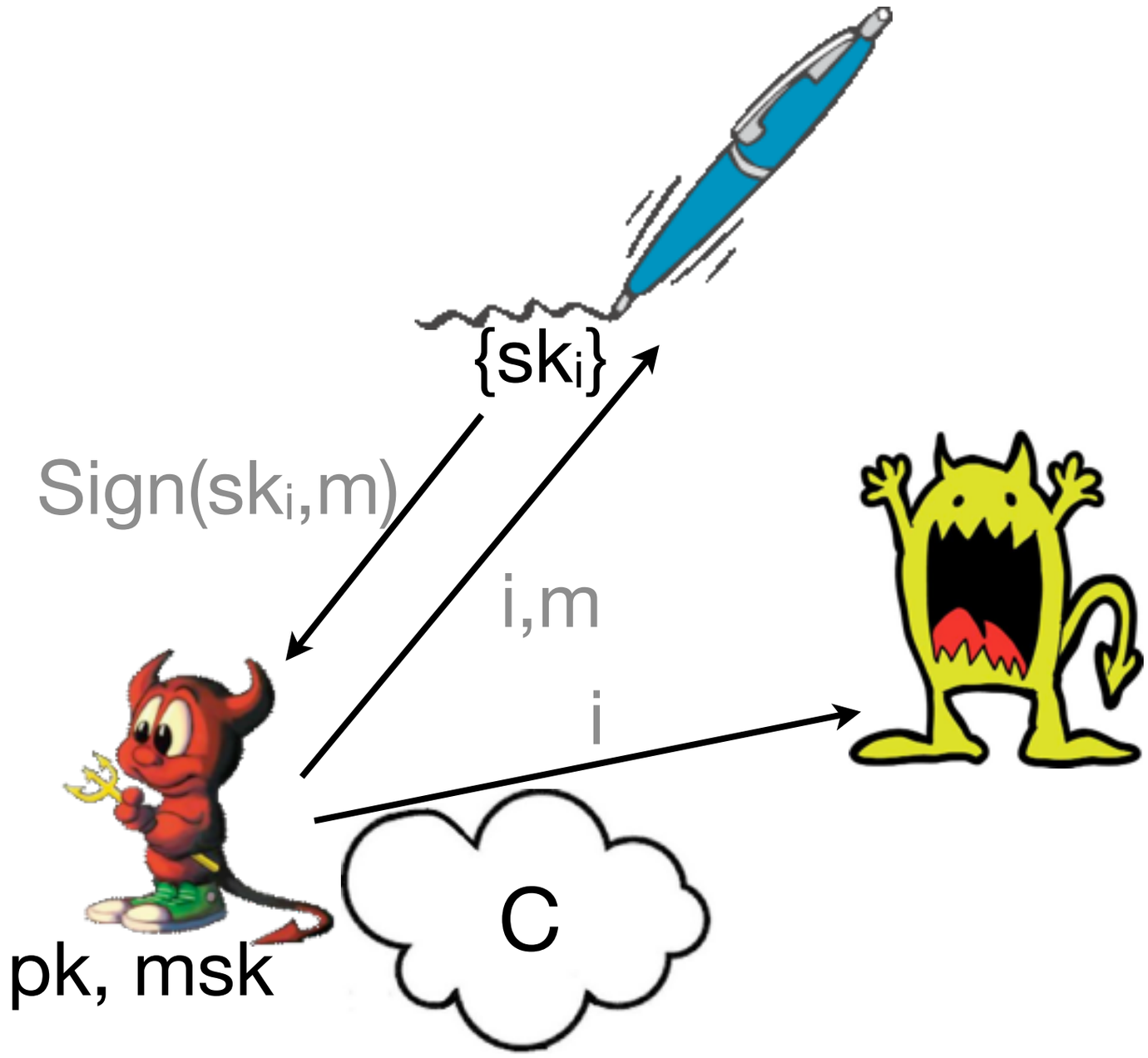
i,m

pk, msk

C

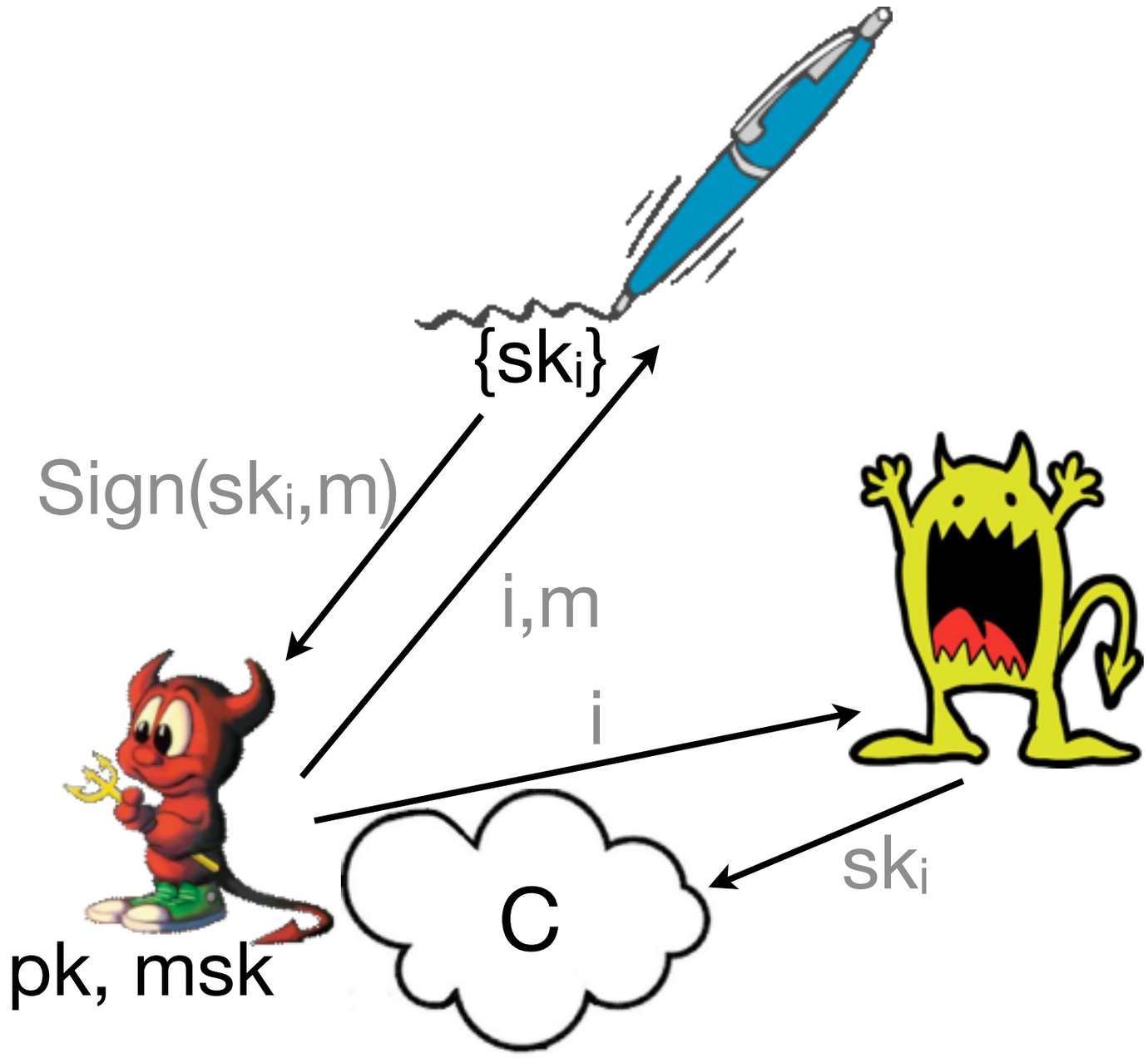
Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition



Phase 1: getting to pick a corrupt coalition

Traceability: a more formal definition



Phase 1: getting to pick a corrupt coalition

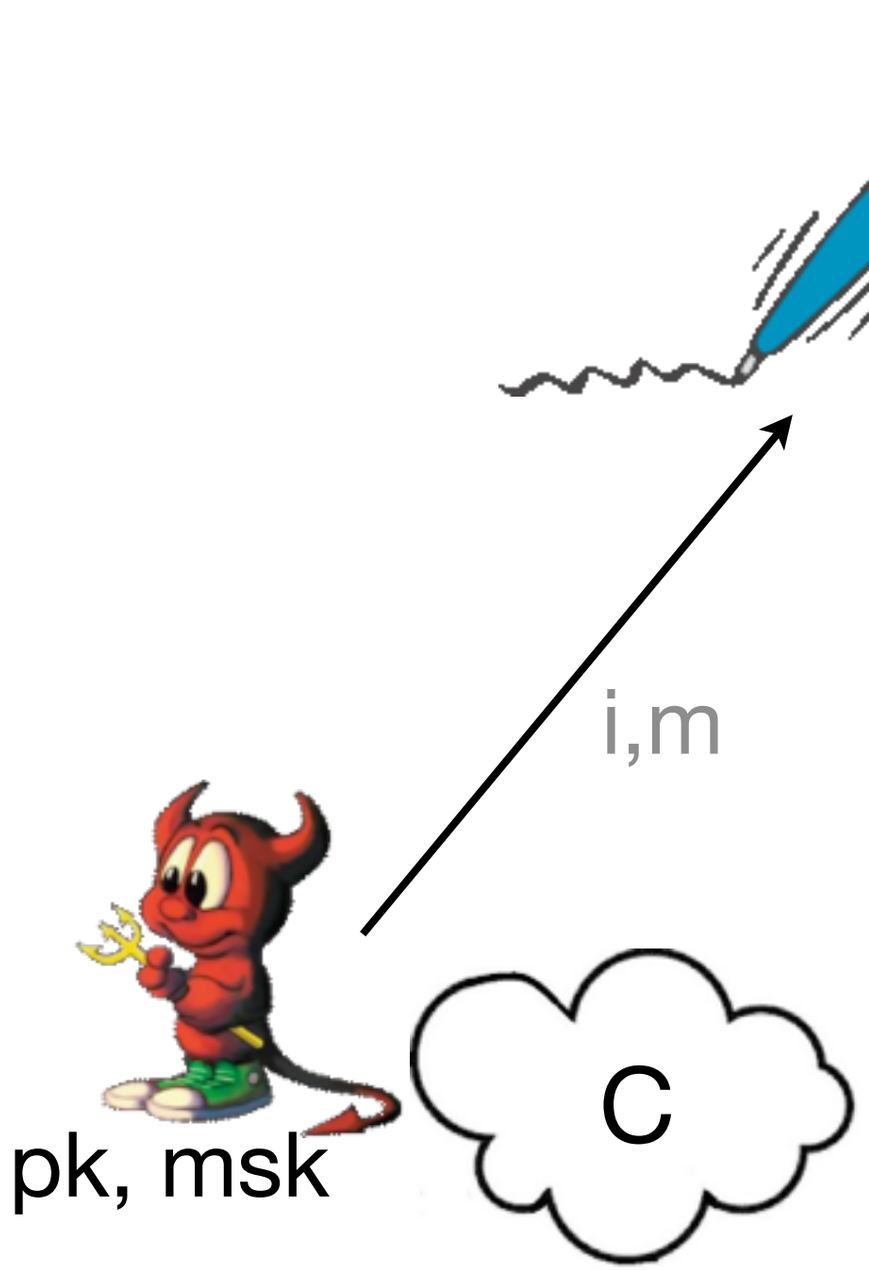
Traceability: a more formal definition

pk, msk

C

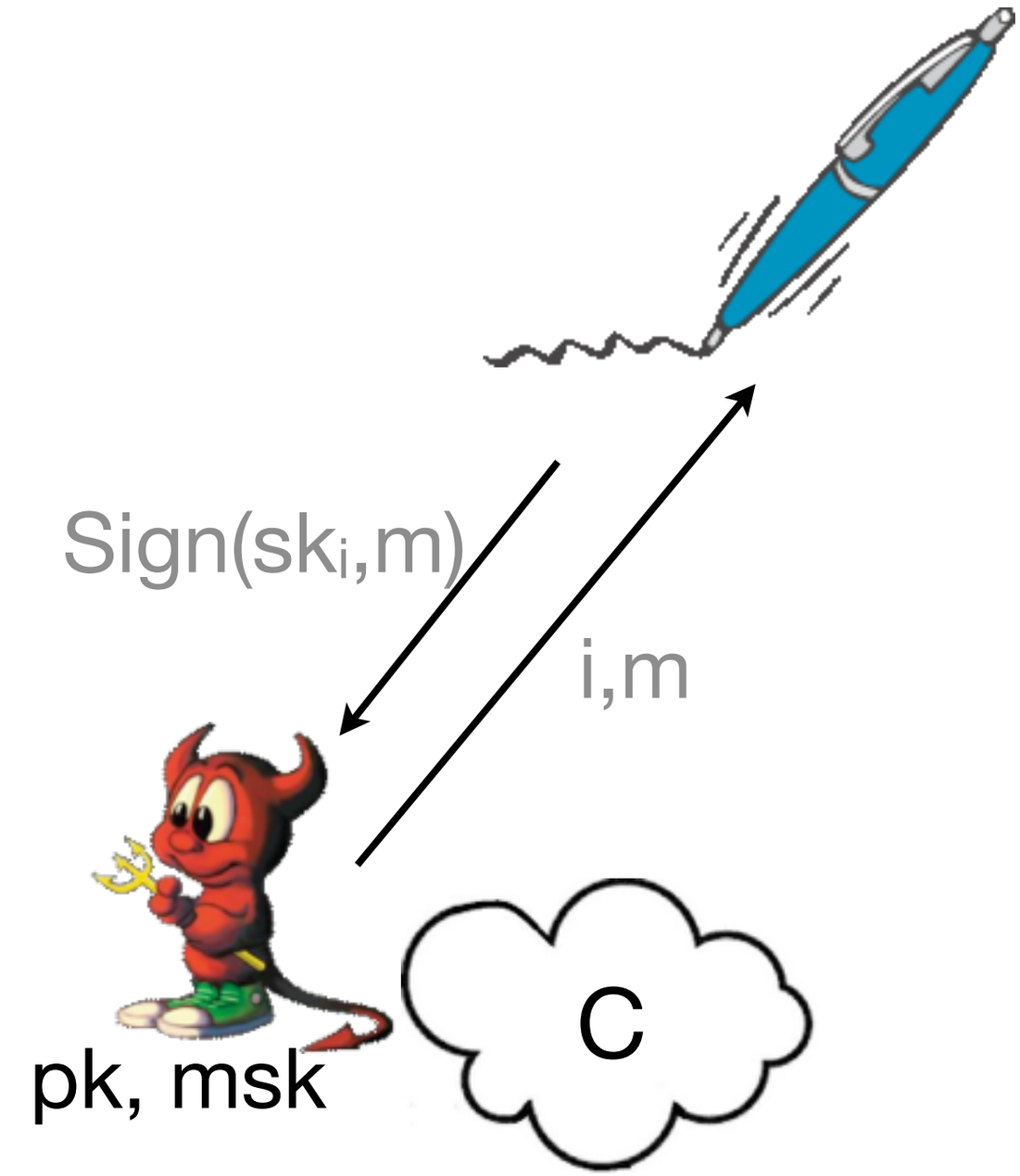
Phase 2: outputting a forgery

Traceability: a more formal definition



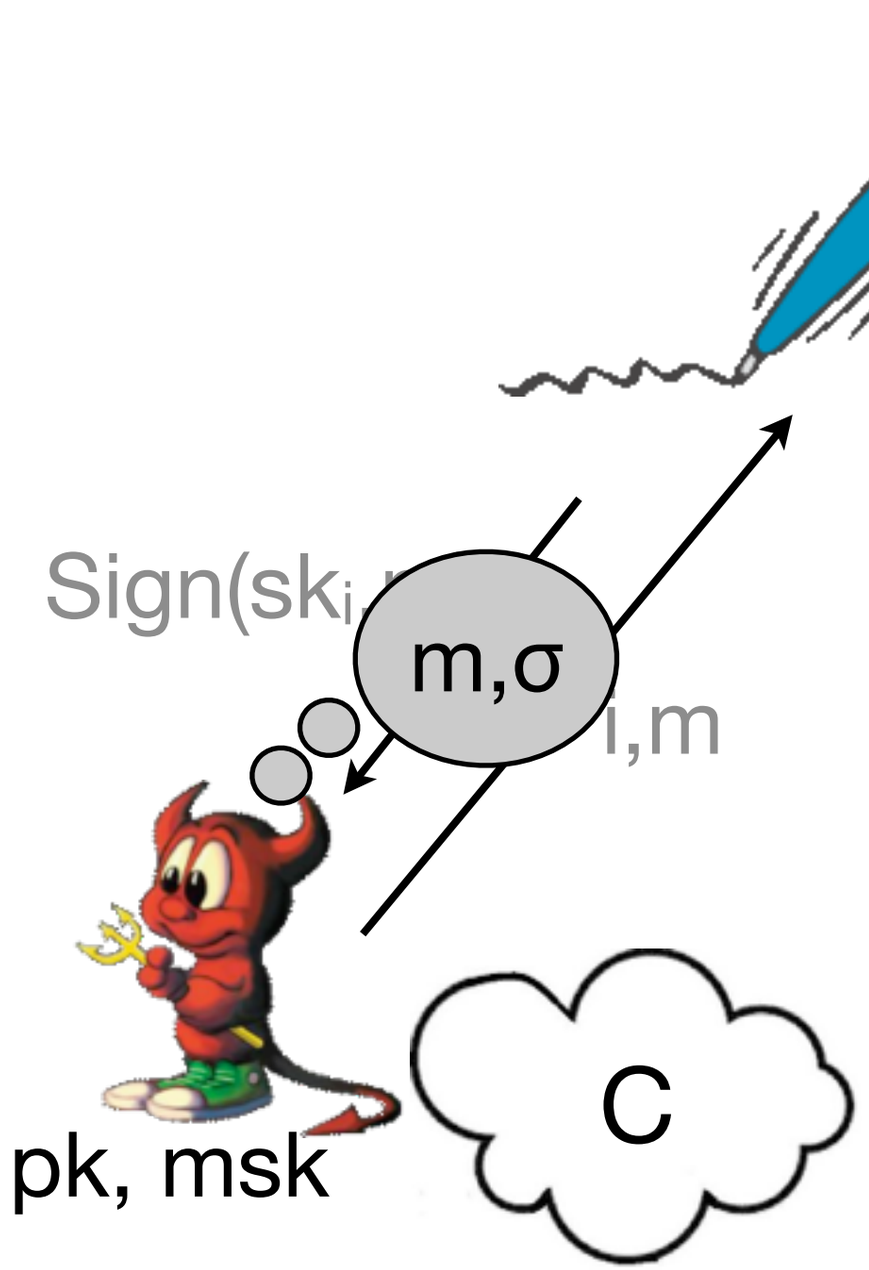
Phase 2: outputting a forgery

Traceability: a more formal definition



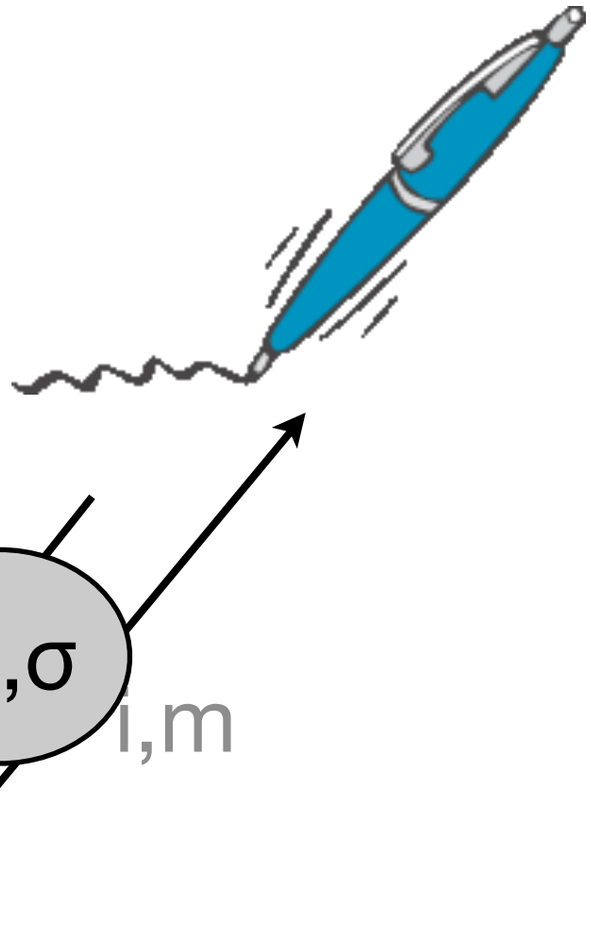
Phase 2: outputting a forgery

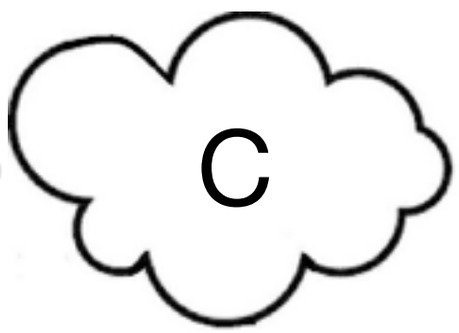
Traceability: a more formal definition



Phase 2: outputting a forgery

Traceability: a more formal definition





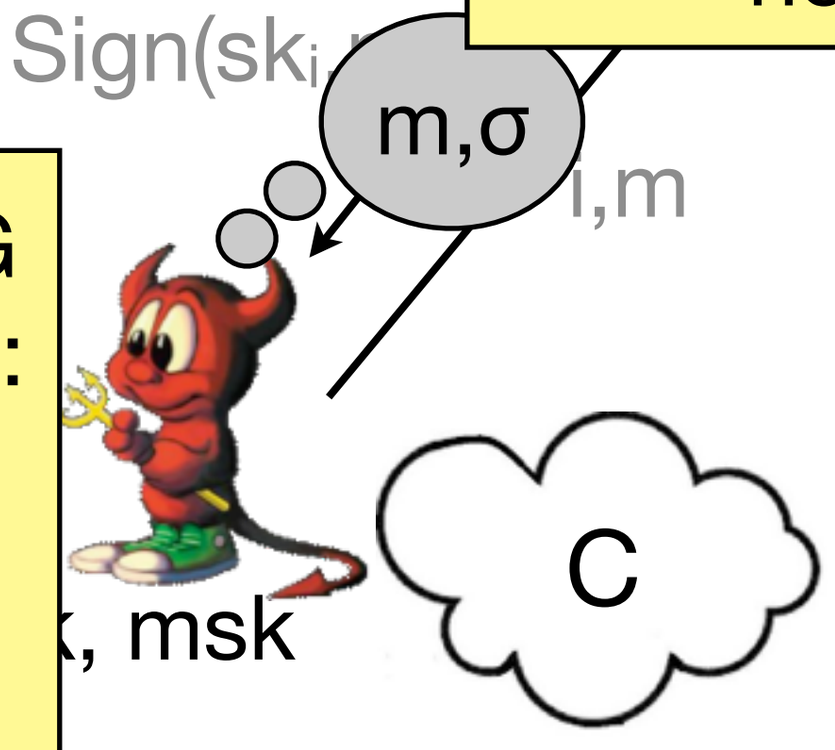
, msk

outputting a forgery

We say that A wins at G if Verify(pk,σ,m) = 1 and: (1) ∃i s.t. Trace(msk,σ,m) = i, (2) i∉C, and (3) A did not query oracle on (i,m)

Traceability: a more formal definition

Say that scheme is traceable if the probability that A wins at G is very small (i.e., negligible)



We say that A wins at G if Verify(pk,σ,m) = 1 and: (1) ∃i s.t. Trace(msk,σ,m) = i, (2) i∉C, and (3) A did not query oracle on (i,m)

Supporting dynamic groups

Back in real-world application: what if someone buys a car?

Supporting dynamic groups

Back in real-world application: what if someone buys a car?

So we can also support dynamic groups in which users join over time

- Replace KeyGen(1^k,1^n) with Setup(1^k) (just outputs msk and pk)
- Add Join() ↔ Enroll(msk) protocol for group master to hand out keys as members join

Supporting dynamic groups

Back in real-world application: what if someone buys a car?

So we can also support dynamic groups in which users join over time

- Replace KeyGen(1^k,1^n) with Setup(1^k) (just outputs msk and pk)
- Add Join() ↔ Enroll(msk) protocol for group master to hand out keys as members join

In practice, this approach could be emulated by a group master who simply runs KeyGen(1^k,1^N) for some N >> n, stockpiles extra keys for later

Using group managers instead of masters

Now, we have group manager who doesn't know your secret key

So Join() ↔ Enroll(msk) is a secure two-party computation at the end of which

the member learns their secret key and nothing else, and the group manager learns nothing (except that the member successfully enrolled)

Using group managers instead of masters

Now, we have group manager who doesn't know your secret key

So Join() ↔ Enroll(msk) is a secure two-party computation at the end of which the member learns their secret key and nothing else, and the group manager learns nothing (except that the member successfully enrolled)

Now it makes sense to split tracing capability, Setup(1^k) will output msk used for enrollment, pk used as group public key, and tk used as tracing key

Using group managers instead of masters

Now, we have group manager who doesn't know your secret key

So Join() ↔ Enroll(msk) is a secure two-party computation at the end of which the member learns their secret key and nothing else, and the group manager learns nothing (except that the member successfully enrolled)

Now it makes sense to split tracing capability, Setup(1^k) will output msk used for enrollment, pk used as group public key, and tk used as tracing key

We can further talk about notions of non-frameability, in which corrupt coalition might also involve the group manager

Supporting revocation

What if someone publishes my secret key on the internet?

Supporting revocation

What if someone publishes my secret key on the internet?

We need a method to revoke member privileges; allow certain members to continue signing on behalf of the group but block others from doing so

Supporting revocation

What if someone publishes my secret key on the internet?

We need a method to revoke member privileges; allow certain members to continue signing on behalf of the group but block others from doing so

This is often accomplished using a revocation list (RL)

- In verifier-local revocation, RL is sent to all verifiers, who then perform some additional checks using Verify(pk,RL,σ,m)
- We could also have remaining signers update their keys to match some updated public key using KeyUpdate(pk′,pk,RL,sk_i) → sk_i'

How do we evaluate group signature schemes?

- Efficiency: want really fast Sign and Verify
- Size of the signatures: want them to be independent of the group size
- Security: want highest level of security (CCA-style anonymity, full traceability)
- Flexibility: group manager? dynamic addition? revocation?
- Uses reasonable assumptions: random oracles? crazy weird-looking assumptions?

Comparison of group signature schemes

	Efficiency	Size	Security	Flexibility	Assumptions	R.O.?
CS'97			CPA-A, PT	manager, +	DLP + strong RSA	
BMW'03		C*	CCA-A, FT	master	TDP	
DKNS'04			CPA-A, FT	manager, +	Strong RSA	
BBS'04			CPA-A, FT	master, -	q-SDH + DLIN	
BSZ'05		C*	CCA-A, FT	master, +	TDP	
BW'06		lg(N)	CPA-A, FT	master, +/-	CDH + SGH	
Groth'06		C*	CCA-A, FT	manager, +	DLIN	
BW'07			CPA-A, FT	master, +/-	CDH + SGH + HSDH	

Comparison of group signature schemes

	Efficiency	Size	Security	Flexibility	Assumptions	R.O.?
CS'97			CPA-A, PT	manager, +	DLP + strong RSA	
BMW'03		C*	CCA-A, FT	master	TDP	
DKNS'04			CPA-A, FT	manager, +	Strong RSA	
BBS'04			CPA-A, FT	master, -	q-SDH + DLIN	
BSZ'05		C*	CCA-A, FT	master, +	TDP	
BW'06		lg(N)	CPA-A, FT	master, +/-	CDH + SGH	
Groth'06		C*	CCA-A, FT	manager, +	DLIN	
BW'07			CPA-A, FT	master, +/-	CDH + SGH + HSDH	

- Holy grail: Efficient, CCA-A and FT secure, fully dynamic but short signatures, secure under mild assumptions and without random oracles

- There's no clear winner here!

Outline

Cryptographic background

Group signatures

Ring signatures

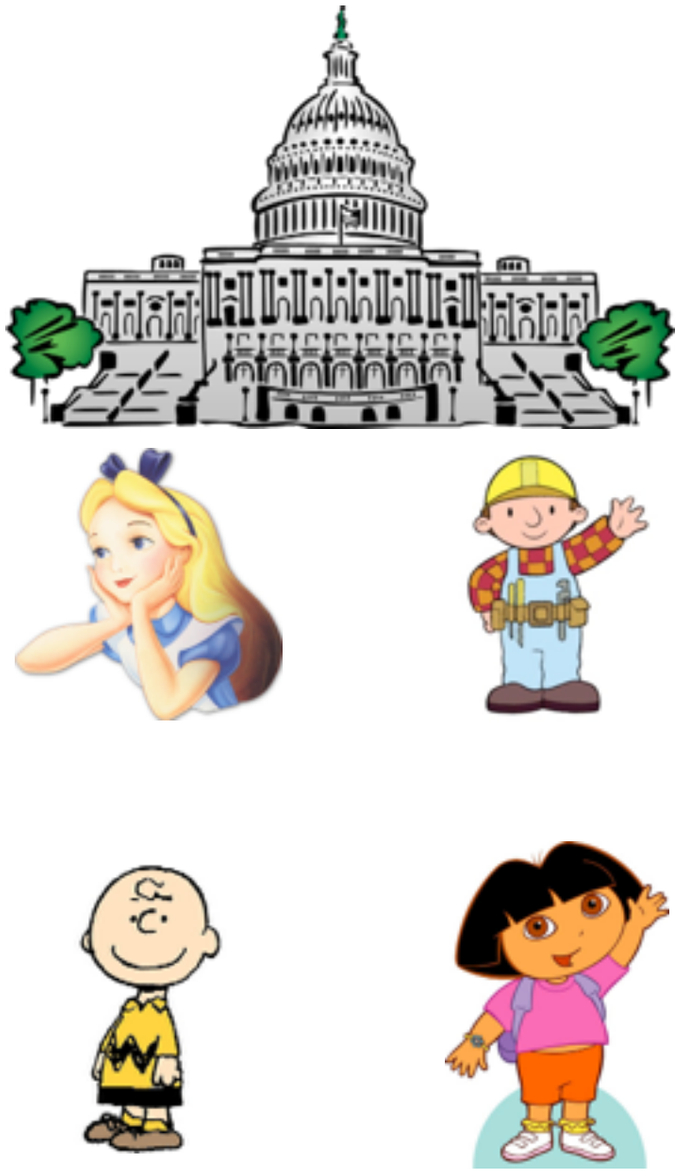
Intuition and motivation

Formal definitions

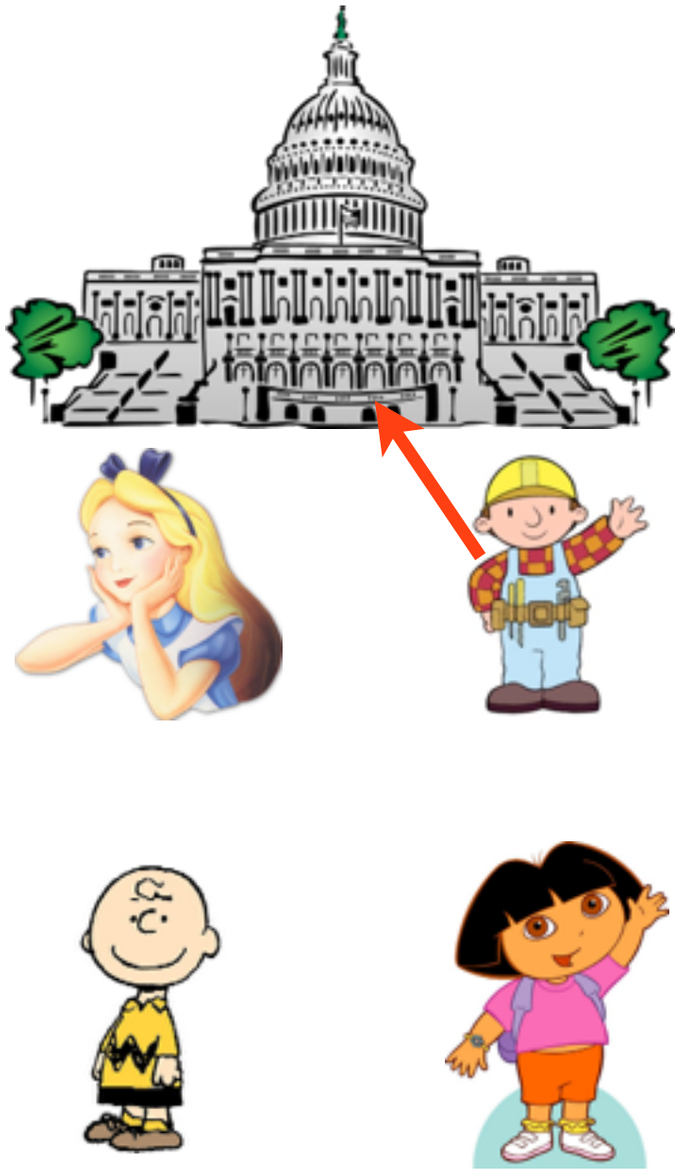
Comparison of existing schemes

Open problems

Ring signatures: why do we want them?

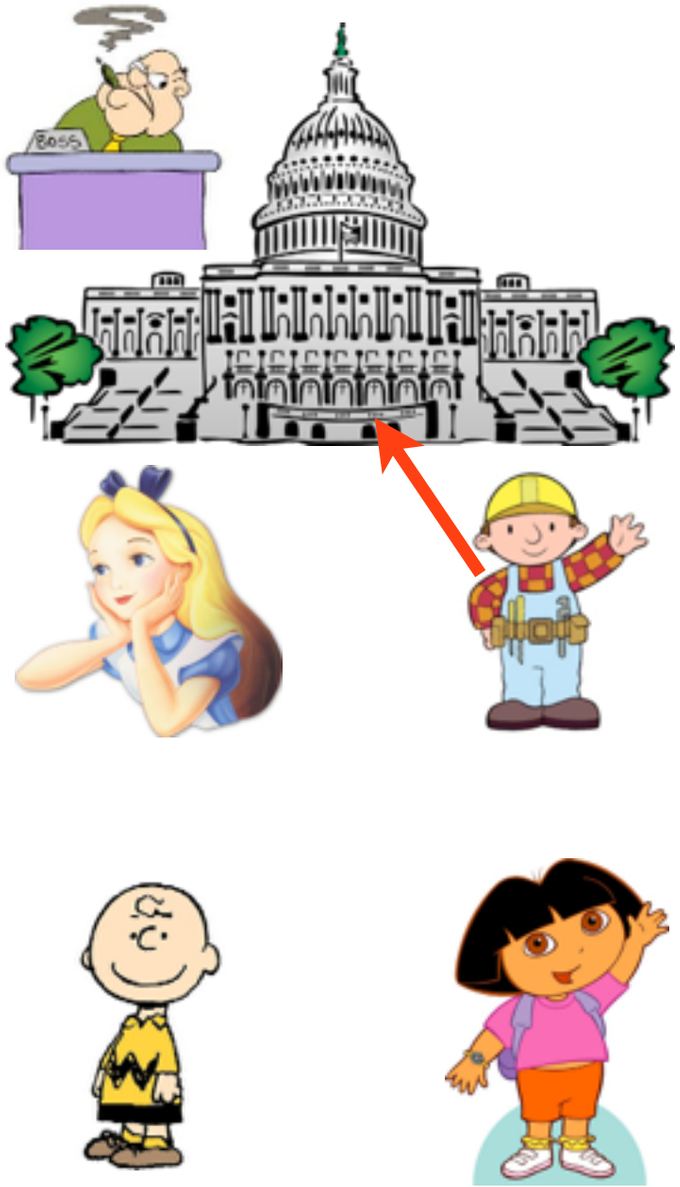


Ring signatures: why do we want them?



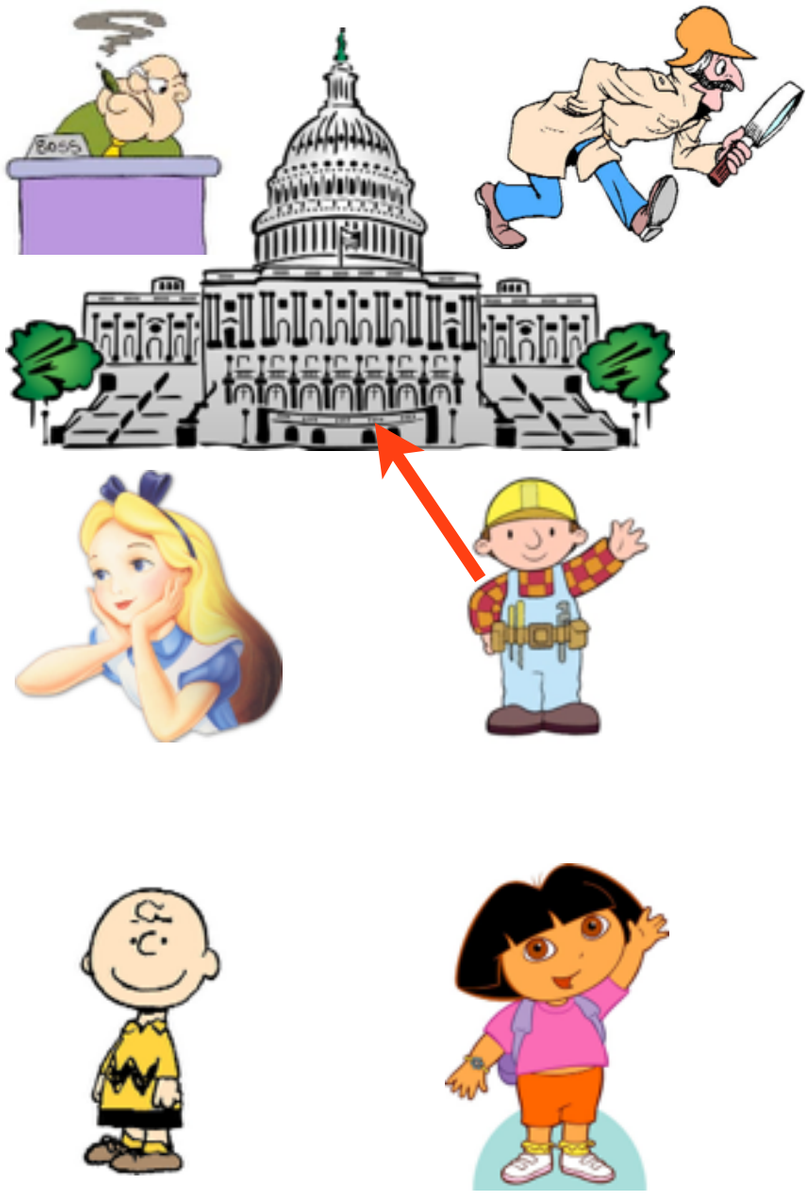
1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

Ring signatures: why do we want them?



1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

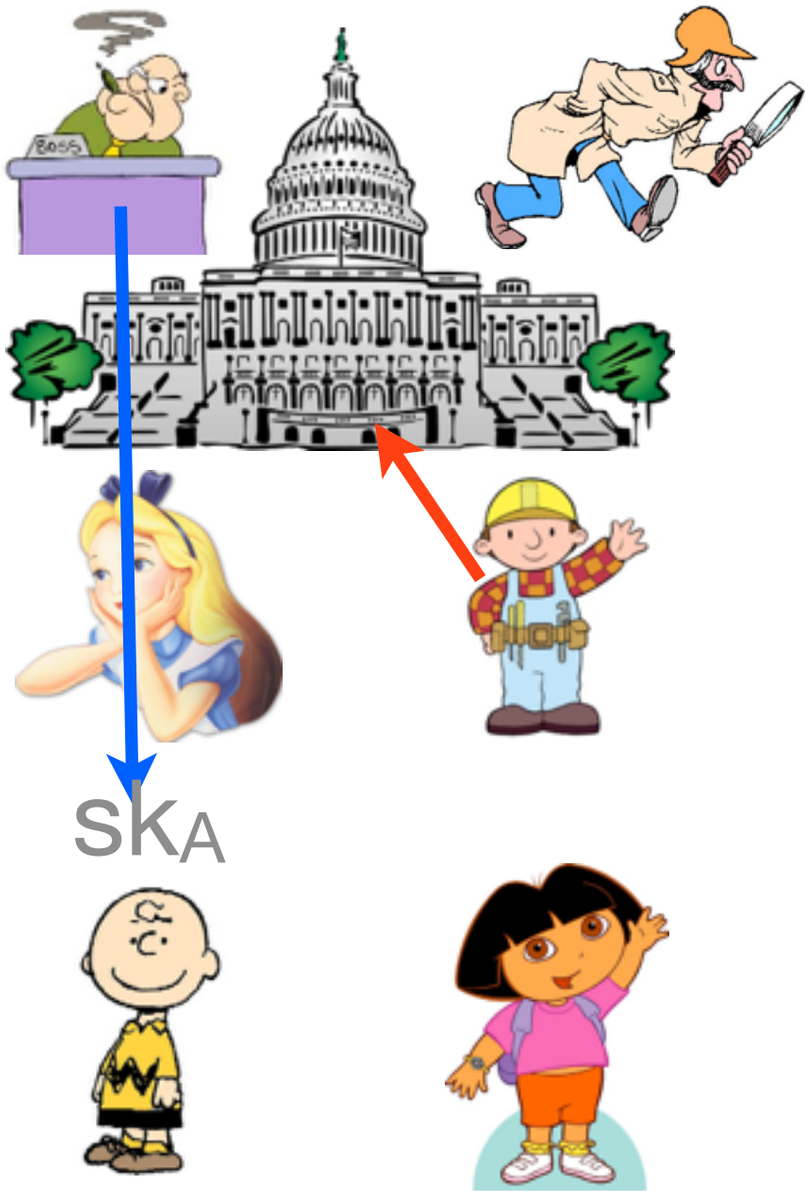
Ring signatures: why do we want them?



1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

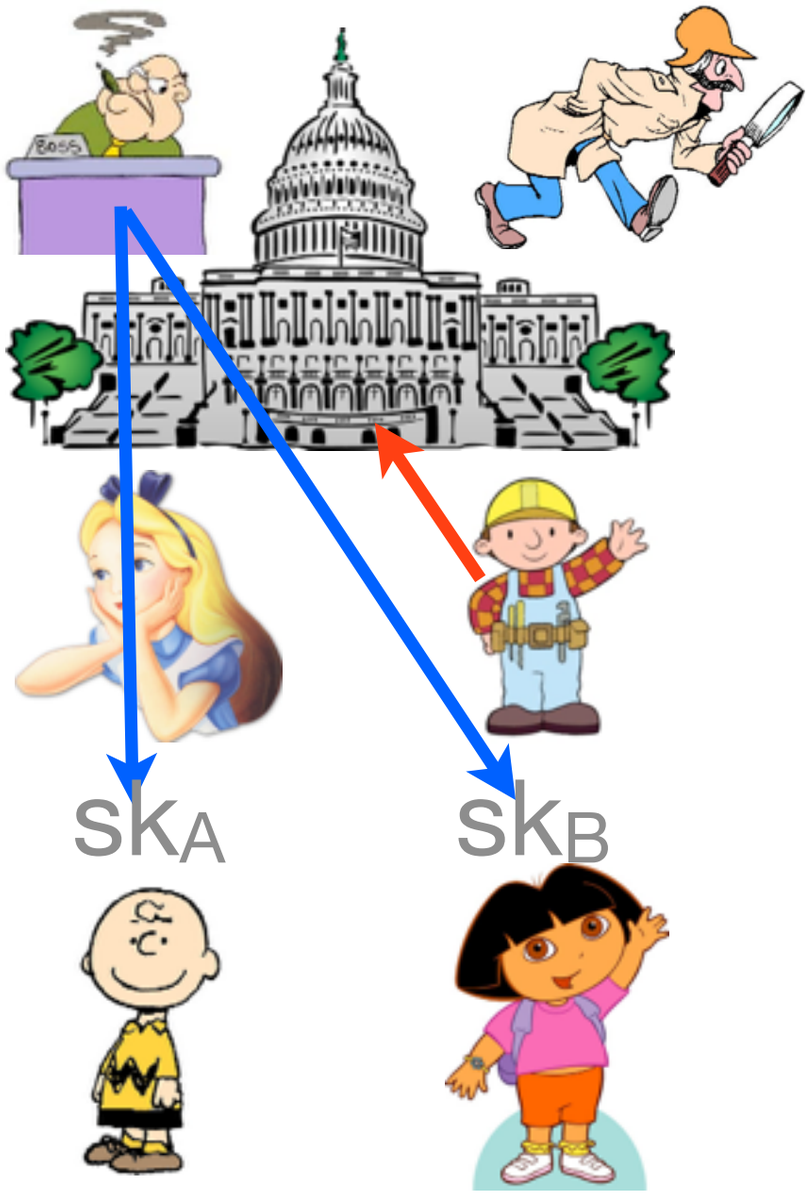
Ring signatures: why do we want them?



1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

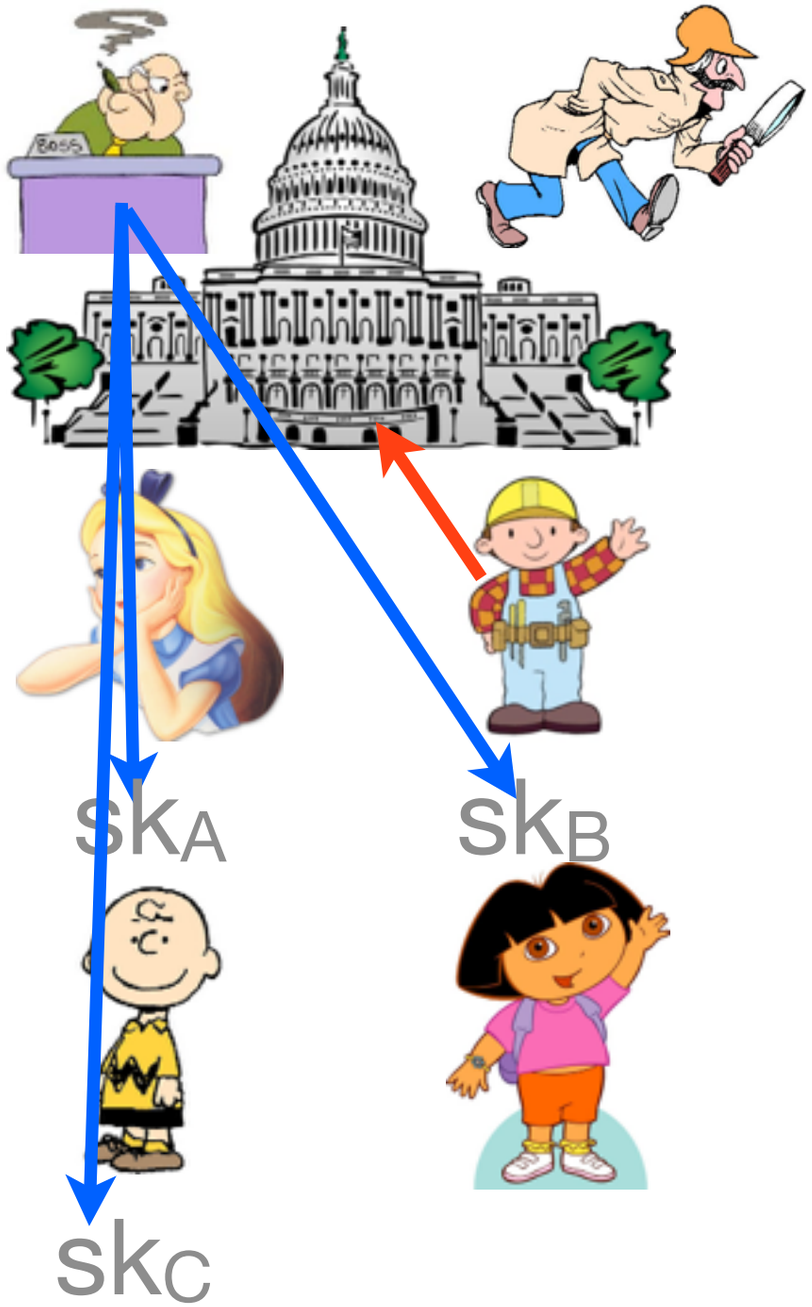
Ring signatures: why do we want them?



1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

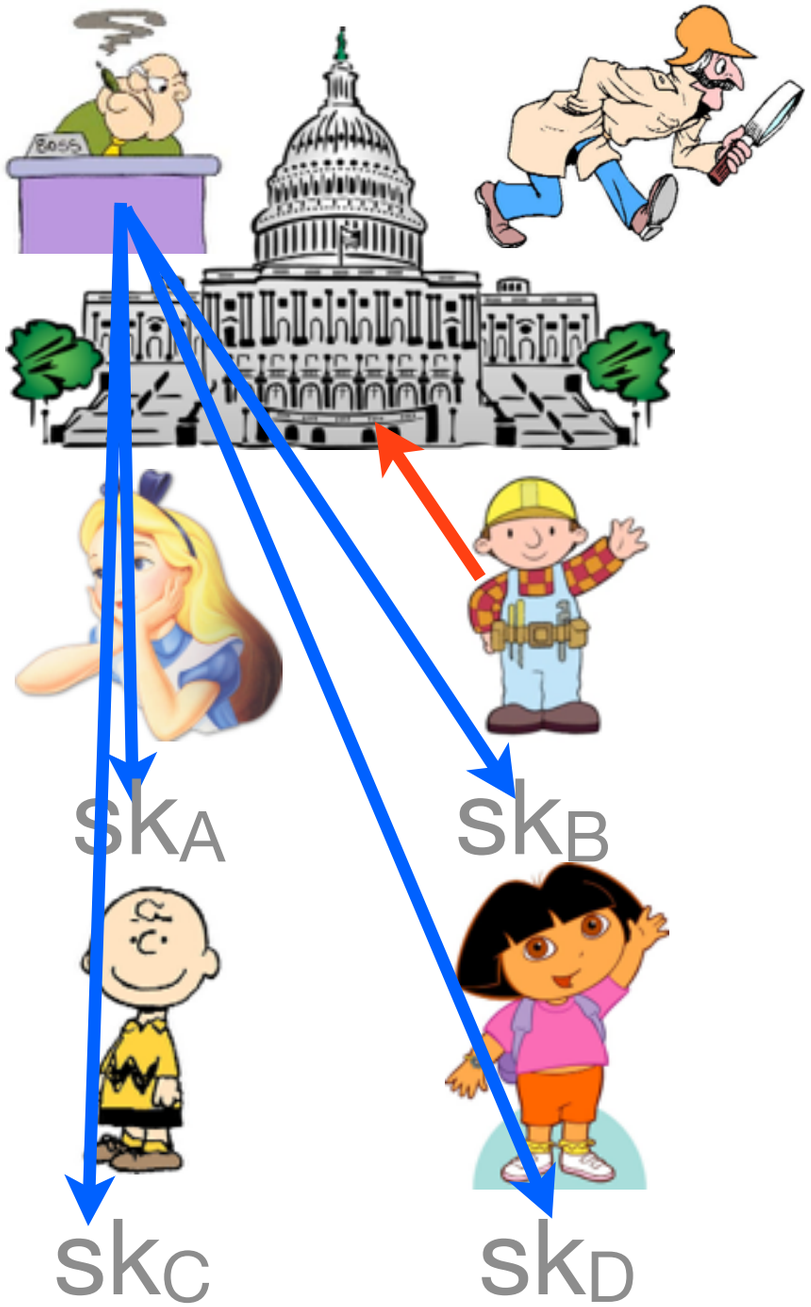
Ring signatures: why do we want them?



1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

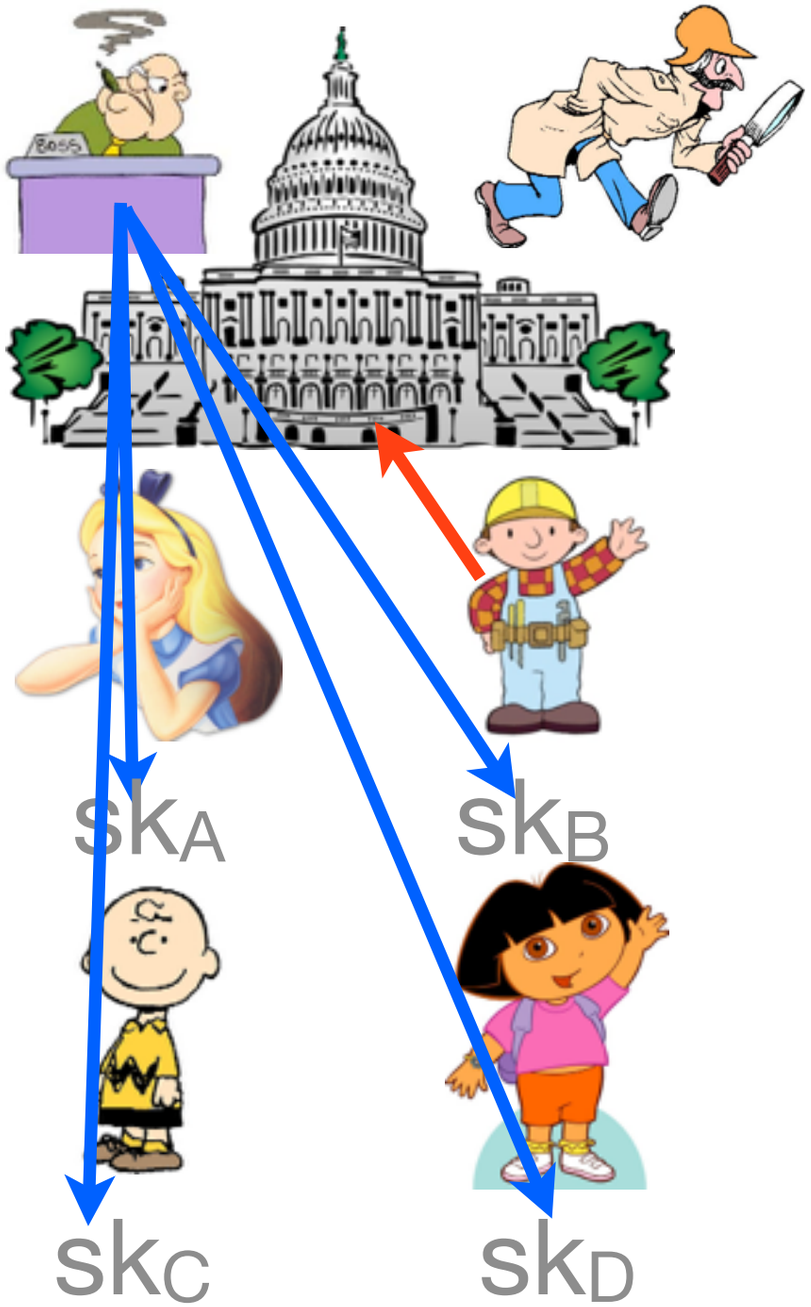
Ring signatures: why do we want them?



1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

Ring signatures: why do we want them?

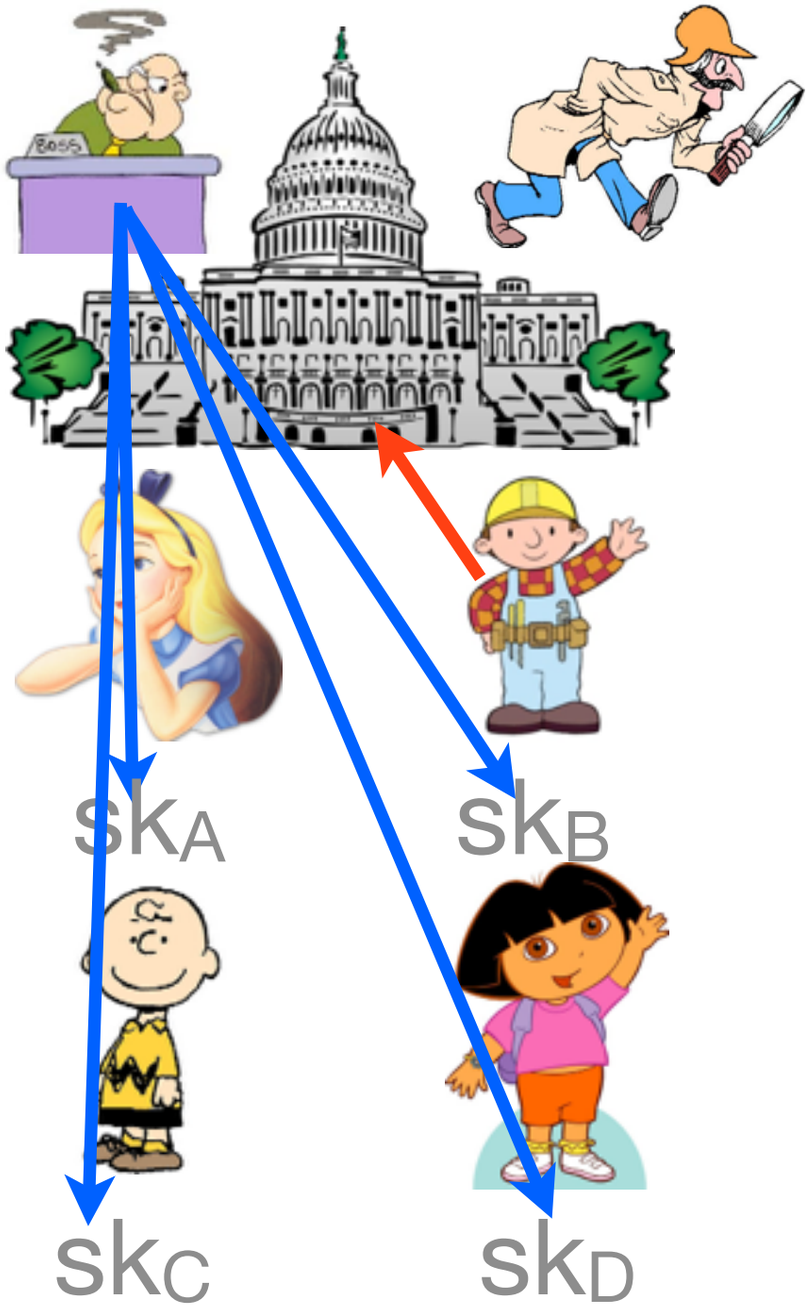


1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

3. Government picks a tracer

Ring signatures: why do we want them?



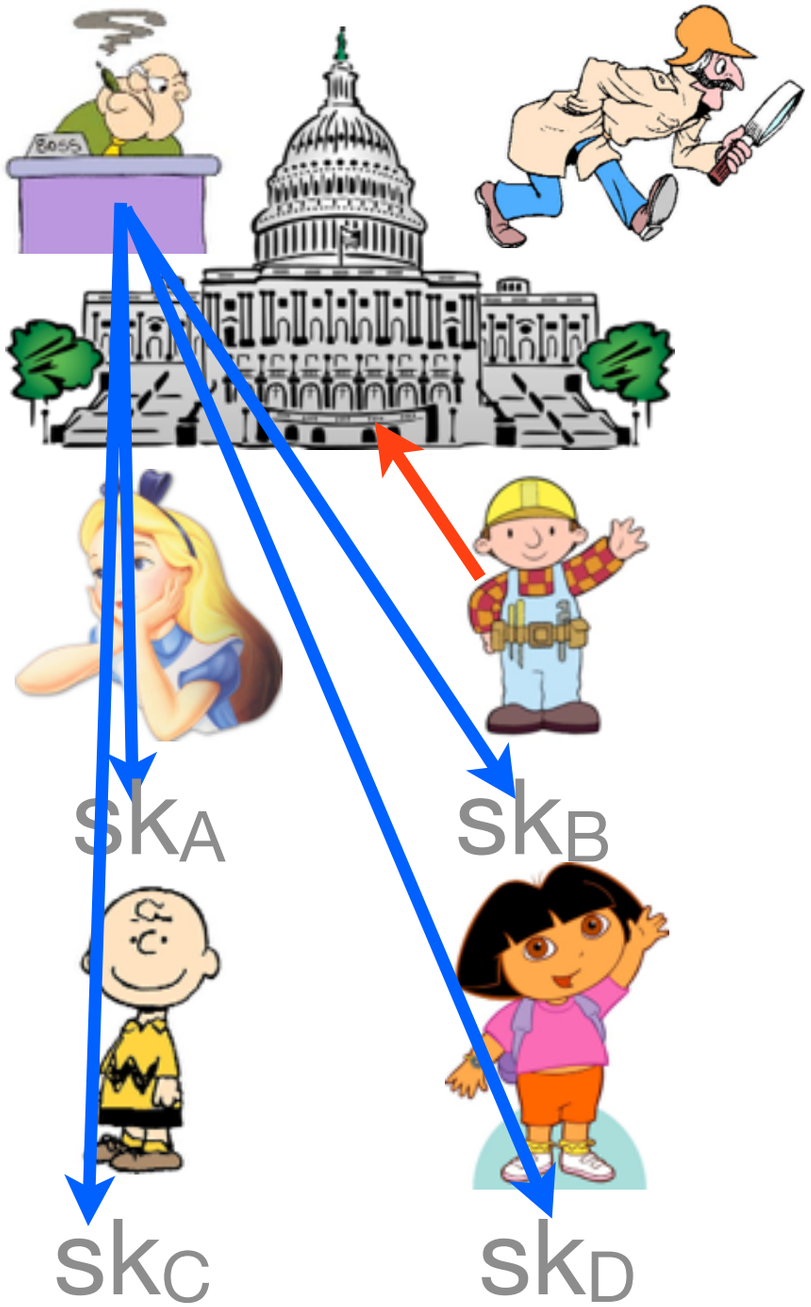
1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

3. Government picks a tracer

4. Boss issues key for Senator #1

Ring signatures: why do we want them?



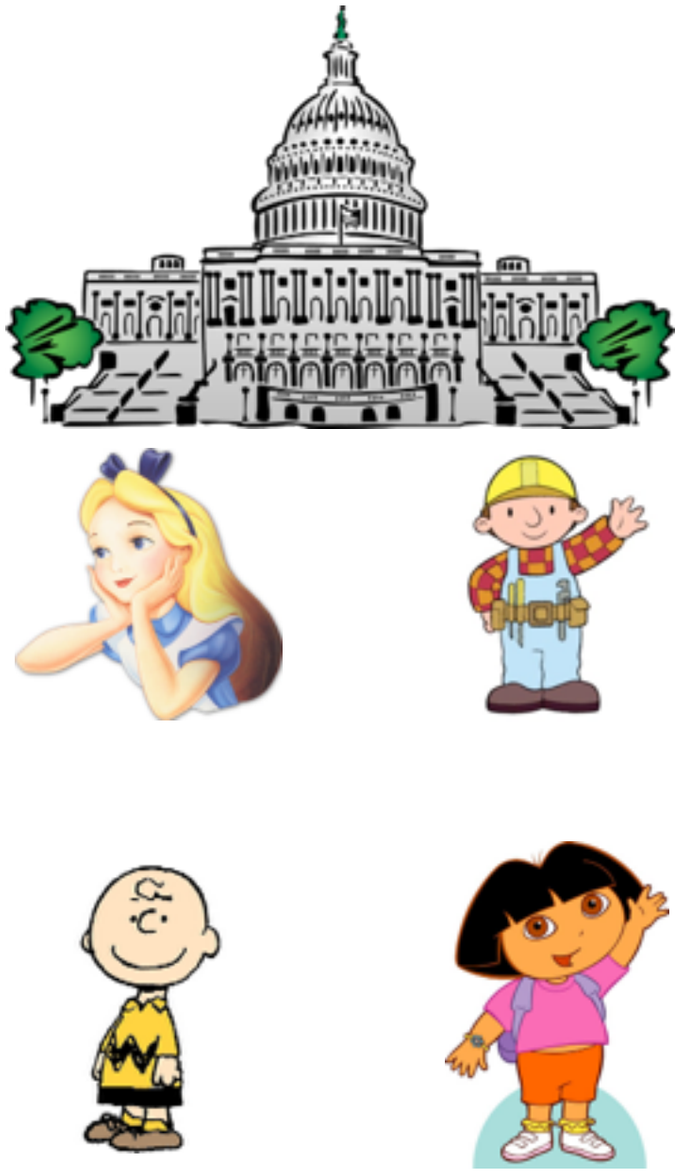
1. Bob contacts the Senate staff, requests that a group be made (for all the senators)

2. Government picks a group master/manager

3. Government picks a tracer ← What if Bob wants to protect his privacy unconditionally?

4. Boss issues key for Senator #1

Ring signatures: why do we want them?



Ring signatures: why do we want them?

pk_A, sk_A pk_B, sk_B

pk_C, sk_C pk_D, sk_D

Ring signatures: why do we want them?

pk_A, sk_A pk_B, sk_B

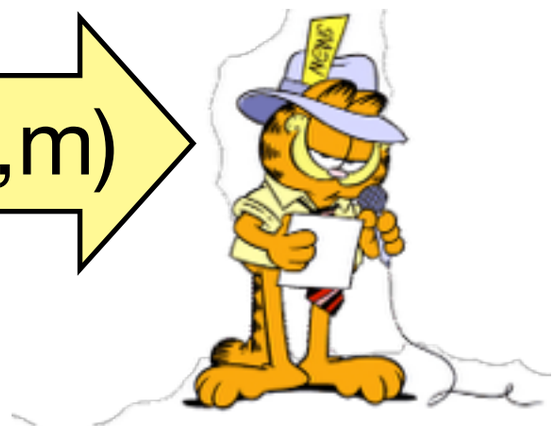
pk_C, sk_C pk_D, sk_D

Ring signatures: why do we want them?

pk_A, sk_A

pk_B, sk_B

$m, \sigma = \mathrm{Sign}(sk_B, R, m)$



pk_C, sk_C

pk_D, sk_D

Ring signatures: why do we want them?

pk_A, sk_A

pk_B, sk_B

pk_C, sk_C

pk_D, sk_D

$m, \sigma = \text{Sign}(sk_B, R, m)$

R = “US senators”

$\{pk_A, pk_B, pk_C, pk_D, \ldots\}$

Ring signatures: why do we want them?

pk_A, sk_A pk_B, sk_B

pk_C, sk_C pk_D, sk_D

$m, \sigma = \text{Sign}(sk_B, R, m)$

Verify(R,σ,m) = 1... so a senator wrote the message...

R = “US senators”

$\{pk_A, pk_B, pk_C, pk_D, \ldots\}$

Ring signatures: why do we want them?

pk_A, sk_A pk_B, sk_B

pk_C, sk_C pk_D, sk_D

$m, \sigma = \text{Sign}(sk_B, R, m)$

Verify(R,σ,m) = 1... so a senator wrote the message... but I don't know if the Senate sanctioned it.

R = "US senators"

$\{pk_A, pk_B, pk_C, pk_D, \ldots\}$

Ring signatures: a formal characterization

- A ring signature is a tuple of algorithms (KeyGen,Sign,Verify)

Ring signatures: a formal characterization

- A ring signature is a tuple of algorithms (KeyGen,Sign,Verify)
- KeyGen(1^k): outputs public key pk and secret key sk

Ring signatures: a formal characterization

- A ring signature is a tuple of algorithms (KeyGen,Sign,Verify)
- KeyGen(1^k): outputs public key pk and secret key sk
- Sign(sk_i,R,m): outputs signature σ on message m

Ring signatures: a formal characterization

- A ring signature is a tuple of algorithms (KeyGen,Sign,Verify)
- KeyGen(1^k): outputs public key pk and secret key sk

- Sign(sk_i,R,m): outputs signature σ on message m

- Verify(R,σ,m): checks that σ is a valid signature on m formed by some member of the ring defined by R (and outputs 1 if yes and 0 if no)

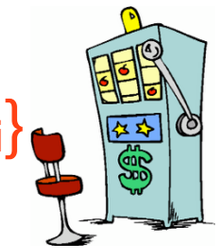
Ring signature anonymity

Anonymity against full key exposure:

Ring signature anonymity

Anonymity against full key exposure:

- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i, sk_i\}$



Ring signature anonymity

Anonymity against full key exposure:

- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i,sk_i\}$

- Phase 2: A gets to see $S=\{pk_i\}$, access signing oracle Sign(.,.,.) that on input (i,R,m) will output Sign(ski,R,m) (we could have $R \not\subset S$)

Ring signature anonymity

Anonymity against full key exposure:

- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i,sk_i\}$

- Phase 2: A gets to see $S=\{pk_i\}$, access signing oracle Sign(.,.,.) that on input (i,R,m) will output Sign(ski,R,m) (we could have $R \not\subset S$)

- Phase 3: A outputs challenge (i_0,i_1,R,m) (again could have $R \not\subset S$) and gets back $Sign(sk_{ib},R,m)$ for some bit b it doesn't know

Ring signature anonymity

Anonymity against full key exposure:

- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i,sk_i\}$

- Phase 2: A gets to see $S=\{pk_i\}$, access signing oracle Sign(.,.,.) that on input (i,R,m) will output Sign(ski,R,m) (we could have $R \not\subset S$)

- Phase 3: A outputs challenge (i_0,i_1,R,m) (again could have $R \not\subset S$) and gets back $Sign(sk_{ib},R,m)$ for some bit b it doesn't know

- Phase 4: A now gets to see all $\{sk_i\}$, eventually outputs a guess bit b′

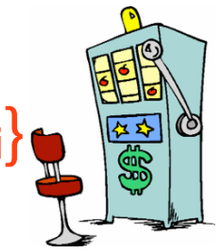
Ring signature unforgeability

We obviously can’t consider traceability, since there is no tracer! So we instead define unforgeability against coalitions and chosen-ring attacks:

Ring signature unforgeability

We obviously can't consider traceability, since there is no tracer! So we instead define unforgeability against coalitions and chosen-ring attacks:

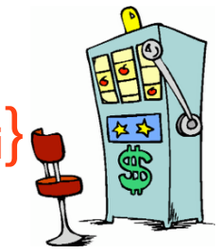
- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i, sk_i\}$



Ring signature unforgeability

We obviously can't consider traceability, since there is no tracer! So we instead define unforgeability against coalitions and chosen-ring attacks:

- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i,sk_i\}$

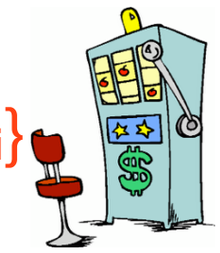


- Phase 2: A gets to see $S=\{pk_i\}$ and has access to two oracles: one that, on input (i,R,m) will output Sign(sk_i,R,m) (we could have $R \not\subset S$), and the other that, on input i, will give A sk_i and consider User i "corrupted"

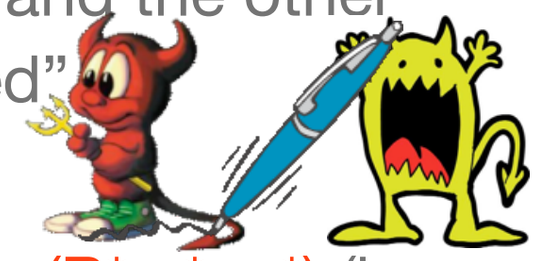
Ring signature unforgeability

We obviously can't consider traceability, since there is no tracer! So we instead define unforgeability against coalitions and chosen-ring attacks:

- Phase 1: KeyGen(1^k) is run m times to get $\{pk_i,sk_i\}$



- Phase 2: A gets to see $S=\{pk_i\}$ and has access to two oracles: one that, on input (i,R,m) will output Sign(sk_i,R,m) (we could have $R \not\subset S$), and the other that, on input i, will give A sk_i and consider User i "corrupted"



- Phase 3: A at some point has to output a successful forgery (R*,σ*,m*) (i.e., such that Verify(R*,σ*,m*) = 1)

How do we evaluate ring signature schemes?

- Efficiency: want really fast Sign and Verify
- Size of the signatures: want them to be independent of the ring size
- Security: want highest level of security (full anonymity, full unforgeability)
- Flexibility: can users pick their own signature schemes?
- Uses reasonable assumptions: random oracles? crazy weird-looking assumptions?

Comparison of ring signature schemes

	Efficiency	Size	Security	Flexibility	Assumptions	R.O.?
RST'01		linear	UFA		TDP	
DKNS'04		C	CFA		Strong RSA	
BKM'06		linear	CFA, FU		TDP	
SW'07		linear	CFA, FU		CDH + SGH	
Boyen'07		linear	UFA, PU		Poly-SDH	

Comparison of ring signature schemes

	Efficiency	Size	Security	Flexibility	Assumptions	R.O.?
RST'01		linear	UFA		TDP	
DKNS'04		C	CFA		Strong RSA	
BKM'06		linear	CFA, FU		TDP	
SW'07		linear	CFA, FU		CDH + SGH	
Boyen'07		linear	UFA, PU		Poly-SDH	

- Holy grail: Efficient, CFA and FU secure, flexible but short signatures, secure under mild assumptions and without random oracles

- Again, there's no clear winner!

Outline

Cryptographic background

Group signatures

Ring signatures

Open problems

Open problems for group signatures

- We already saw this “holy grail” of a scheme that is efficient, CCA-A and FT secure, fully dynamic but short signatures, secure under mild assumptions and without random oracles

Open problems for group signatures

- We already saw this “holy grail” of a scheme that is efficient, CCA-A and FT secure, fully dynamic but short signatures, secure under mild assumptions and without random oracles

- Also would be nice to see more applications in the real world (just DAA and VSC for now)

Open problems for group signatures

- We already saw this “holy grail” of a scheme that is efficient, CCA-A and FT secure, fully dynamic but short signatures, secure under mild assumptions and without random oracles

- Also would be nice to see more applications in the real world (just DAA and VSC for now)

- Generic construction for a fully dynamic scheme (i.e., one that supports revocation)

Open problems for group signatures

- We already saw this “holy grail” of a scheme that is efficient, CCA-A and FT secure, fully dynamic but short signatures, secure under mild assumptions and without random oracles

- Also would be nice to see more applications in the real world (just DAA and VSC for now)

- Generic construction for a fully dynamic scheme (i.e., one that supports revocation)

- Better definitions and formalizations for revocation

Open problems for ring signatures

- **Find a real-world application!!**

Open problems for ring signatures

- **Find a real-world application!!**

- Again, achieve holy grail of scheme that is efficient, CFA and FU secure, flexible but short signatures, secure under mild assumptions and without random oracles

Open problems for ring signatures

- **Find a real-world application!!**

- Again, achieve holy grail of scheme that is efficient, CFA and FU secure, flexible but short signatures, secure under mild assumptions and without random oracles

- Figure out way to overcome this linear-sized signature barrier (ideally without random oracles)

Open problems for ring signatures

- **Find a real-world application!!**

- Again, achieve holy grail of scheme that is efficient, CFA and FU secure, flexible but short signatures, secure under mild assumptions and without random oracles

- Figure out way to overcome this linear-sized signature barrier (ideally without random oracles)

- Can we even achieve flexibility using a non-generic construction?

Open problems for ring signatures

- **Find a real-world application!!**

- Again, achieve holy grail of scheme that is efficient, CFA and FU secure, flexible but short si[illegible]ns and without random oracles

- Figure out way to c[illegible]rrier (ideally without random oracles)

- Can we even achie[illegible]ruction?

Any questions?