Modern Cryptography: Lecture 10

The Public Key Revolution II/II

Daniel Slamanig

AUSTRIAN INSTITUTE OF TECHNOLOGY

Organizational

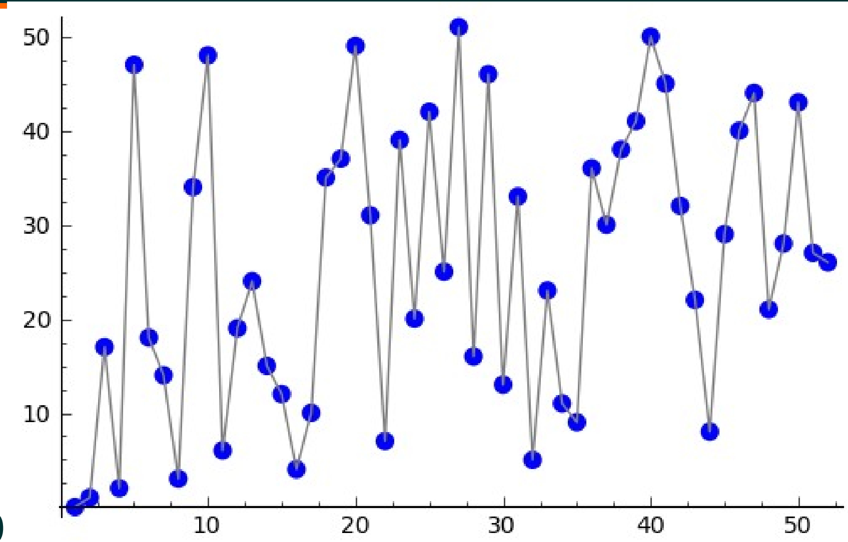
- Where to find the slides and homework?
  - https://danielslamanig.info/ModernCrypto18.html
- How to contact me?
  - daniel.slamanig@ait.ac.at
- Tutor: Karen Klein
  - karen.klein@ist.ac.at

- Official page at TU, Location etc.
  - https://tiss.tuwien.ac.at/course/courseDetails.xhtml?dswid=8632&dsrid=679&courseNr=192062&semester=2018W
- Tutorial, TU site
  - https://tiss.tuwien.ac.at/course/courseAnnouncement.xhtml?dswid=5209&dsrid=341&courseNumber=192063&courseSemester=2018W

- Exam for the second part: Thursday 31.01.2019 15:00-17:00 (Tutorial slot)
  - No tutorial this week → exam for first part

Discrete Logarithms

- We consider a cyclic group G of order q with generator g, so $G = \{g^0, \ldots, g^{q-1}\}$
- The DL problem: given $h=g^x$ to find the unique $x \in \mathbb{Z}_q$
- Let $\mathcal{G}$ be a group generator that on input 1n outputs a description of a cyclic group (G, q, g) with $\|q\|$=n (binary length)



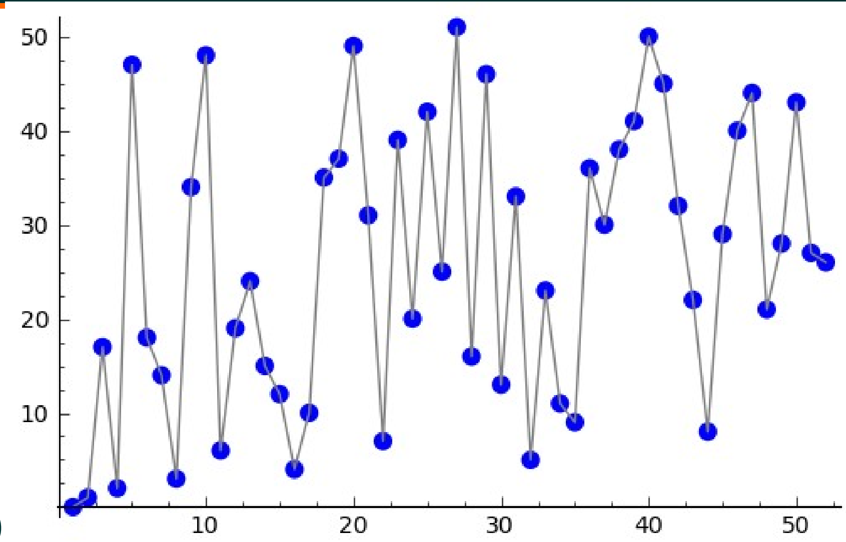
$\mathbb{Z}^*_{53}$, g=2

The discrete-logarithm experiment $\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n)$:

1. Run $\mathcal{G}(1^n)$ to obtain (G, q, g), where G is a cyclic group of order q (with $\|q\|$ = n), and g is a generator of G.
2. Choose a uniform $h \in G$.
3. $\mathcal{A}$ is given G, q, g, h, and outputs $x \in \mathbb{Z}_q$.
4. The output of the experiment is defined to be 1 if $g^x = h$, and 0 otherwise.

Discrete Logarithms

- We consider a cyclic group G of order q with generator g, so G = $\{g^0, \ldots, g^{q-1}\}$
- The DL problem: given $h=g^x$ to find the unique $x \in \mathbb{Z}_q$
- Let $\mathcal{G}$ be a group generator that on input 1n outputs a description of a cyclic group (G, q, g) with $\|q\|$=n (binary length)



$\mathbb{Z}^*_{53}$, g=2

The discrete-logarithm experiment $\mathrm{DLog}_{\mathcal{A},\mathcal{G}}(n)$:

1. Run $\mathcal{G}(1^n)$ to obtain (G, q, g), where G is a cyclic group of order q (with $\|q\|$ = n), and g is a generator of G.

2. Choose a uniform $h \in G$.

DEFINITION 8.62 We say that the discrete-logarithm problem is hard relative to $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathrm{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1] \le \mathrm{negl}(n).$$

Problems Related to the DLOG Problem

- We will now take a look at two problems related but weaker than the DLP; the computational (CDH) and the decisional Diffie–Hellman (DDH) problem
- Let $\mathbf{DH}_g(h_1, h_2) := g^{\log_g h_1 \cdot \log_g h_2}$
  - If $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$, then $\mathbf{DH}_g(h_1, h_2) = g^{x_1 x_2} = h_1^{x_2} = h_2^{x_1}$

- CDH Problem
  - Given (G, q, g, h_1, h_2) compute $\mathbf{DH}_g(h_1, h_2)$

DEFINITION: We say that the CDH problem is hard relative to $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there is a negligible function negl such that

$$\Pr[\mathcal{A}(G, q, g, g^x, g^y) = g^{xy}] \leq \text{negl}(n),$$

where the probabilities are taken over the experiment in which $\mathcal{G}(1^n)$ outputs (G, q, g), and then uniform $x, y\ \mathbb{Z}_q$ are chosen.

Problems Related to the DLOG Problem

- DDH Problem
  - Given (G, q, g) and uniform random $h_1, h_2 \in G$, distinguish $\mathbf{DH}_g(h_1, h_2)$ from uniformly random $h' \in G$

DEFINITION 8.63: We say that the DDH problem is hard relative to $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there is a negligible function negl such that

$$\Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] \leq \mathrm{negl}(n),$$

where in each case the probabilities are taken over the experiment in which $\mathcal{G}(1^n)$ outputs (G, q, g), and then uniform $x, y, z \in \mathbb{Z}_q$ are chosen.

Clearly, if we can solve DL, then we can solve DDH and CDH

DDH is a stronger assumption than CDH (HW)

There are groups where the CDH is assumed hard, but the DDH is easy (HW)

Algorithms for Computing Discrete Logarithms

- Two types of algorithms
  - Generic ones: apply to arbitrary groups
  - Specific ones: tailored to work for some specifc class of groups

Generic for groups of order q:

-Baby step/giant step (Shanks)*: $\mathcal{O}(\sqrt{q} \cdot \text{polylog}(q))$ time and $\mathcal{O}(\sqrt{q})$ space
-Pollard's rho*: $\mathcal{O}(\sqrt{q} \cdot \text{polylog}(q))$ time and constant space

Generic for groups of order q (if factorization is known/easy to compute):

-Pohlig-Hellman: Reduces to finding DL in group or order q' with q' the largest prime dividing q (use then any algorithm to solve the DL)

Specific algorithm for $\mathbb{Z}^*_p$:

-Index Calculus/Number Field Sieve: Subexponential with runtime $2^{\mathcal{O}((\log p)\cdot(\log\log p))}$

* time complexity optimal for generic algorithms

The Baby-Step/Giant-Step Algorithm I/II

- Want to solve DL problem for some $h=g^x$ in (G, q, g)
- We know that h must lie somwhere in the cycle $\{g^0, \ldots, g^{q-1}\}$
  - Computing all elements would take $\Omega(q)$ time!

- Take some elements of the cycle at steps $t=\lfloor\sqrt{q}\rfloor$ (the "giant steps")
  - Gives us a list $(g^0, g^t, g^{2t}, \ldots, g^{\lfloor q/t\rfloor \cdot t})$ with gaps of at most t elements
  - We know h lies in one of the gaps
  - Compute a list $(h \cdot g^1, \ldots, h \cdot g^t)$ of shifts of h (the "baby steps")
  - One of the points in the "baby list" will be equal to one in the "giant list", i.e., $h \cdot g^i = g^{k \cdot t}$ for some i and k
  - And determine $x = (kt - i) \bmod q$

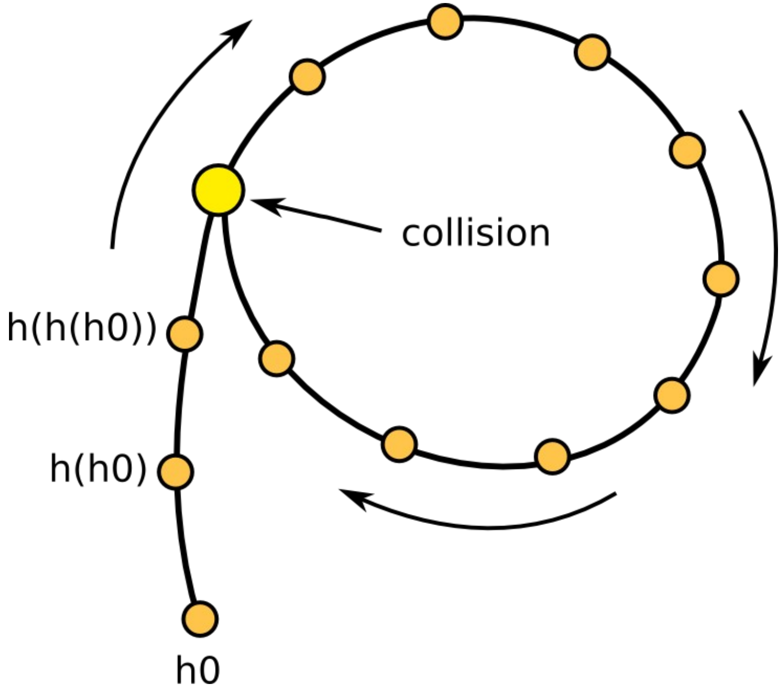
The Baby-Step/Giant-Step Algorithm II/II

- Complexity
  - $\mathcal{O}(\sqrt{q})$ exponentiations/multiplications
  - Sorting the "giant list" takes $\mathcal{O}(\sqrt{q} \cdot \log q)$
  - Binary search for each element from "baby list" in $\mathcal{O}(\log q)$
  - Overall $\mathcal{O}(\sqrt{q} \cdot \text{polylog}(q))$ time <u>but need to store</u> $\mathcal{O}(\sqrt{q})$ elements

- Can we do better generically?

The Pollard Rho Algorithm*

- Idea: Let $H_{g,h}$: $\mathbb{Z}_q \times \mathbb{Z}_q \rightarrow G$ be defined by $H_{g,h}(x_1 , x_2) = g^{x_1} \cdot h^{x_2}$
- The birthday bound says we find a collision in $H_{g,h}$ in time $\mathcal{O}(\sqrt{q})$
- Is possible with constant memory (see §5.4.2)
- If $H_{g,h}(x_1 , x_2) = H_{g,h}(x_1' , x_2')$ with $x_1 \neq x_1$ and $x_2 \neq x_2$ then solve $\gamma(x_2 - x_2') = (x_1' - x_1) \bmod q$ for γ



- Some issues not yet considerd
  - Range of hash function must be subset of its domain: Use a standard cryptographic hash function F: $G \rightarrow \mathbb{Z}_q \times \mathbb{Z}_q$ to obtain the input for G

* we use the description from the book for consistency

Choice of Discrete Logarithm Hard Groups

- Generic vs. special algorithms
  - If only generic algorithms are available parameters can be chosen much smaller; Yields more efficient group operations
- Prime order vs. composite order groups
  - Prime order: Discrete logarithm problem is hardest in prime order groups and finding generators is trivial
  - Composite order: Need to have subgroup of sufficient size (recall: largest prime dividing the order; may need to consider specific algorithms). Finding generators is more cumbersome.
- Prime order groups are preferable (there are some more reasons why discussed later, see also HW)

Choice of Discrete Logarithm Hard Groups

- Groups that are of interest
  - $\mathbb{Z}^*_p$ (does not have prime order)
  - Prime order q subgroups of $\mathbb{Z}^*_p$
  - Elliptic curve groups

What about $\mathbb{Z}_p$ with addition?

Effective Key Length	**RSA** Modulus Length	**Discrete Logarithm** Order-q Subgroup of $\mathbb{Z}_p^*$	**Discrete Logarithm** Elliptic-Curve Group Order q
112	2048	p: 2048, q: 224	224
128	3072	p: 3072, q: 256	256
192	7680	p: 7680, q: 384	384
256	15360	p: 15360, q: 512	512

Key sizes recommended by NIST (from §9.3)

Prime Order Subgroups of $\mathbb{Z}^*_p$

- We can "craft" p in a way that it has a prime order q subgroup of desired size

<u>THEOREM 8.64</u> Let $p = rq + 1$ with p, q prime. Then

$$G = \{h^r \bmod p \mid h \in \mathbb{Z}^*_p\}$$

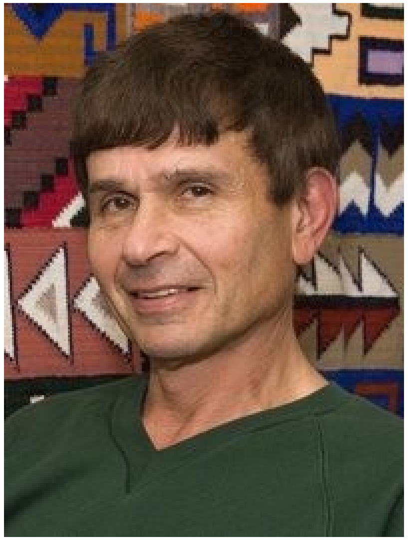
is a subgroup of $\mathbb{Z}^*_p$ of order q.

p is called safe prime if r=2

- Choosing uniform element in G?
  - Choose random h from $\mathbb{Z}^*_p$ and compute $h^r \bmod p$
- Determine if given h is in G (any $h \neq 1$ that is in G is a generator)
  - Check if $h^q = 1 \bmod p$

p and q need to be chosen such that the running time of the NFS (depends on the length of p), **and** the running time of generic algorithms (depends on the length of q) **will be approximately equal**.

Elliptic Curves



Neal Koblitz: **Elliptic Curve Cryptosystems**. Mathematics of Computation, AMS, 1987.

Victor S. Miller: **Use of Elliptic Curves in Cryptography**. Advances in Cryptology – CRYPTO '85

- Groups discussed so far <u>directly</u> rely on modular arithmetic
- Why not use different groups? Elliptic curve groups?
  - Only generic algorithms for the DLP known!

Rationale: "it is extremely unlikely that an index calculus attack on the elliptic curve method will ever be able to work" [Miller, 85]

What are Elliptic Curves?

- An elliptic curve E over a field (we only condsider $\mathbb{F}_p$ with $p \geq 5$, and in particular large p) is a cubic equation

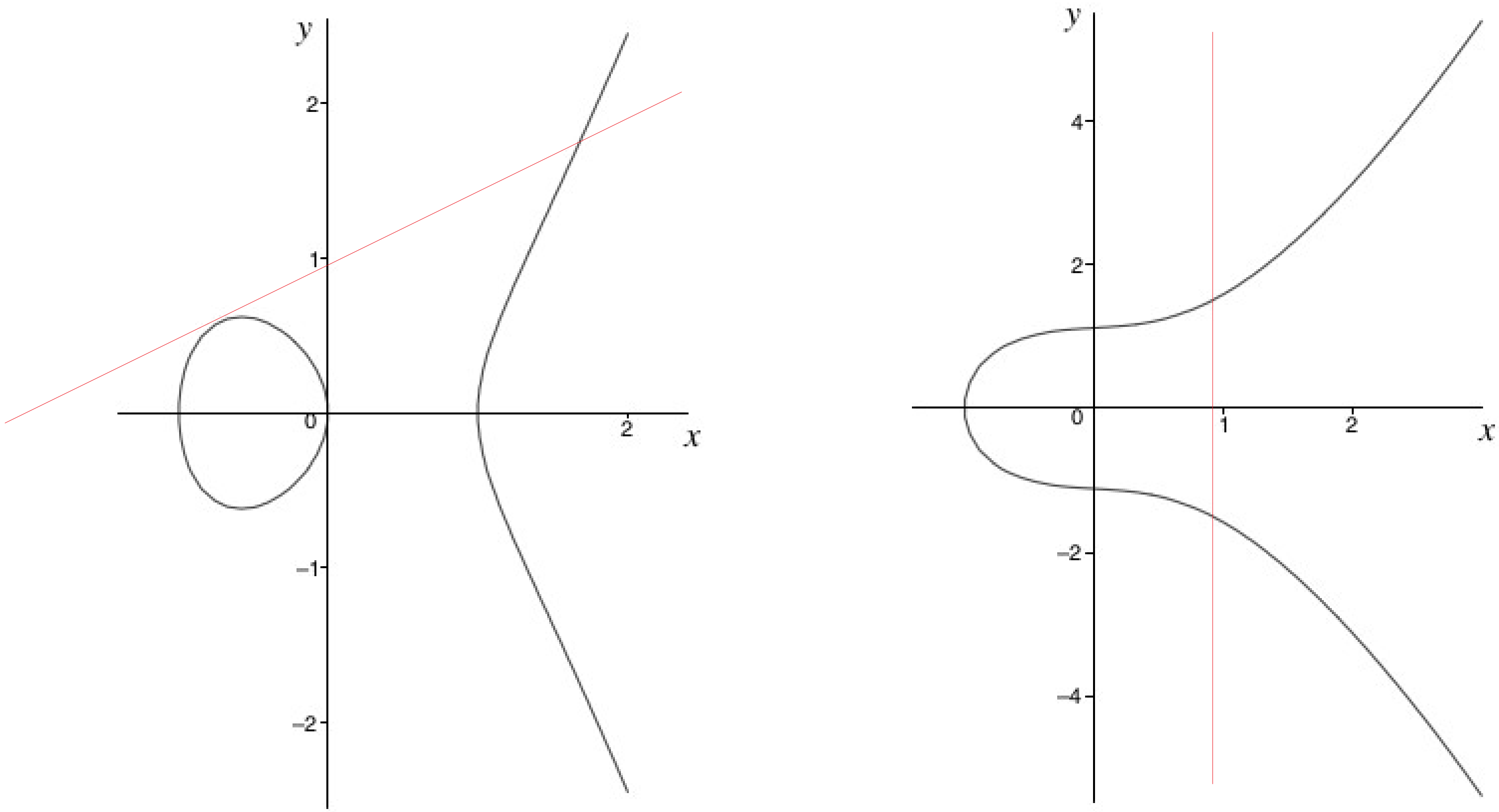
$$y^2 = x^3 + ax + b \qquad \text{(short Weierstrass equation)}$$

  with $a, b \in \mathbb{Z}_p$ and $-16(4a^3 + 27b^2) \neq 0 \bmod p$ (the curve is "smooth")

- Let $E(\mathbb{Z}_p) = \{(x, y) \mid x, y \in \mathbb{Z}_p \text{ and } y^2 = x^3 + ax + b \bmod p\} \cup \{\mathcal{O}\}$
  - The elements in $E(\mathbb{Z}_p)$ are called the points on the elliptic curve E
  - $\mathcal{O}$ is called the point at infinity (it will act as the identiy)

Elliptic Curves over the Reals

A useful way to think about $E(\mathbb{Z}_p)$ is to look at the graph over the reals



(a) $E_1 : y^2 = x^3 - x$ (b) $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

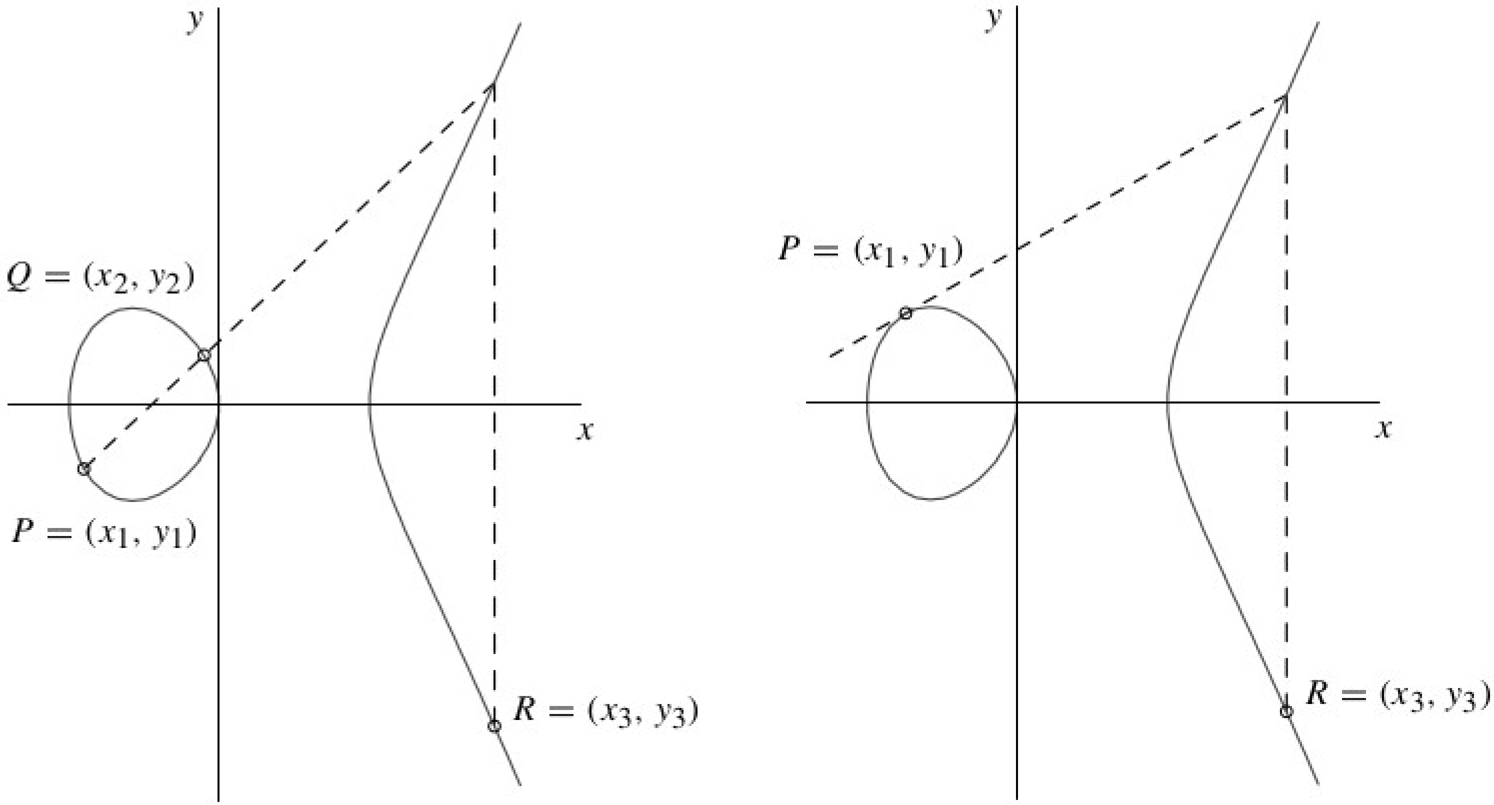
We can think of the point at infinity of sitting on top of the y-axis and lying on every vertical line

Every line intersecting the curve intersects in exactly three points

- Point P is counted twice if line is tangent to the curve
- Point at infinity is counted when the line is vertical

Elliptic Curves: Group Law ("chord-and-tangent rule")

- $E(\mathbb{Z}_p)$ forms a group with additive identity $\mathcal{O}$
  - $\mathcal{O} + P = P + \mathcal{O} = P$ for all $P \in E(\mathbb{Z}_p)$
  - If $P = (x, y) \in E(\mathbb{Z}_p)$, then $(x, y) + (x, -y) = \mathcal{O}$ and $-\mathcal{O} = \mathcal{O}$



(a) Addition: $P + Q = R$.

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{and} \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

(b) Doubling: $P + P = R$.

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{and} \quad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Elliptic Curves

- For cryptographic applications and in particular for the DLP to be hard we need (sub-) groups of large prime order.
- How large are these elliptic curve groups?
  - Let us define a quadratic residue (QR): An element $y \in \mathbb{Z}^*_p$ is a quadratic residue modulo p if there is an $x \in \mathbb{Z}^*_p$ such that $x^2 = y \bmod p$.
  - For $p > 2$ prime, half the elements in $\mathbb{Z}^*_p$ are QRs, and every QR has exactly two square roots.
  - If we look at the equation $y^2 = x^3 + ax + b$, each RHS value that is a QR yields two points on the curve and if RHS is 0 it yields one
  - So we heuristically expect to find expect to find $2 \cdot (p-1)/2 + 1 = p$ points + the point of infinitey, i.e., p+1 points.

THEOREM 8.70 (Hasse bound): Let p be prime, and let E be an elliptic curve over $\mathbb{Z}_p$. Then $p + 1 - 2\sqrt{p} \leq |E(\mathbb{Z}_p)| \leq p + 1 + 2\sqrt{p}$.

Elliptic Curves

- How to find curves?
  - We could just randomly generate them: But for random curves the group order will be "close" to uniformly distributed in the Hasse interval
  - We also need to exclude weak curves, i.e., elliptic-curve groups over $\mathbb{Z}^*{}_p$ whose order is equal to p (anomalous curves) or p+1 (supersingular curves), etc.
  - There are efficient algorithms for counting points on curves, efficiently generating curves
- Typically we use pre-computed standardized curves
  - Standards for Efficient Cryptogrpahy (SEC)
  - National Institute of Standards and Technology (NIST)
  - ECC Brainpool (RFC 5639)
  - Curve25519, Curve448
  - Or BN or BLS if they need to be pairing-friendly

Elliptic Curves

- Now if we have a suitable elliptic curve group $E(\mathbb{Z}_p)$ (or a subgroup) of large prime order q generated by P, we can define the set {1P, ..., qP}
- We can define the elliptic curve DLP (ECDLP) as given Q=xP to compute $x \in \mathbb{Z}_q$
  - Analogously we can define CDH and DDH
- We can use our efficient square-and-multiply algorithm and apply it to this setting (double-and-add) to compute the scalar multiplication efficiently

Elliptic Curves

- Although curves standardized decades ago are still widely used, there happened a lot in the last decades
- Starting with Kocher'99, side-channel attacks and their counter-measures have become extremely sophisticated
- Decades of new research yielding faster, simpler and safer ways to do ECC
- Suspicion surrounding previous standards: Snowden leaks, dual EC-DRBG backdoor, etc., lead to conjectured weaknesses in the NIST curves
- Other specific classes of curves enable secure cryptographic pairings
  - and thus interesting applications such as practical identity- and attribute-based cryptography (see Guest Lecture)

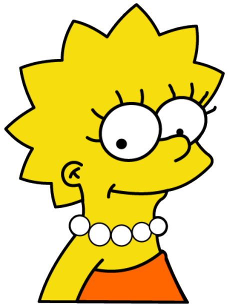
Back to Key Exchange Protocols

Example: KE in $\mathbb{Z}^*_p$ (128 bit security – p: 3072 bit)

p =

5809605995369958062859502533304574370686975176362895236661486152287203730997110225737336044533118407251326157754980517443990529594540047121662885672187
0324010321116397064404988440498509890516272002447658070418123947296805400241048279765843693815222923612087790447698927432257517380769795688113095791255
1133309324351955378481630638158016186020024749256844815024251530444957718760413642873858099017255157393414625583036640591500086964373205321856683254529
1107903722831634138599586406690325959725187447169059540805012310209639011750748760017095360734234945757416272994856013308616958529958304677637019181594
0885283450612858638982717634572948835466388795543116154464463301992543823400162920570907511755338881619189872955915315366987012922676854655174379157908
2315484463478026010289171803249539607504189948551381112697730747896907485704371071615012131592202455675924123901315291971095646840637944291494161435710
7914462567329693649

g = 123456789



g^a mod p =

1974966481832271932862620186142505559719097997625337606540081479948757754456670542185781051331382174972068905995549284294506678994768546685955940340934936375624510789382969603134886961788481424913516872530546022029662470461057707715772483216821171742461283211956785376315202786494034647973536919967369935770926871783856022988735589541210564305228996197614537270822178234757462238037900142350513967990494465082246618501681499574014746384567166244019067013944724470150525694177463721850933025357393837919800705723814217290296516393042343612687649717077634843006689239728687091216655686698309786578047401579166115635085698868474877726766712073860961529476071145597063402090591037030181826355218987380945462945580355697525966763466146993277420884712557411847558661178122098955149524361601993365326052422101474898256696660124195726100495725510022002932814218768060112310763455404567248761396399633344901857872119208518550803791724

a =

714768716640595718790536055473965826924051861459165223549126157152970971006791700379049243301160194978810890876961315928313863262109512949445844004974889298038584931918128447572321023987160439062006177648318875457556233770853912505292364631833219121732146413465584525491722837877275669558984521996220294508922696650742652691278024464164009025927104004338958261141986237587898819361218794559180286406267986483957813927304368495559776413009721221824915810964579376354556655462988377785956808915788215112735742204226463791705999176775673042069842239249481690677789617492307207129760345580262107210922054662739697748553543758990879608882627763290293452560094576029847391361388767554386622479265299978059886472414530462194527618119899746477252908878060493179541951463829228890455778045929437305265410485180264002079415193983851143425084273119820368274789460587100304977477069244278989689910572120963577252034804024499138445834 48

b =

6554562094646949336068268581603170496942310472762446825117743874970612887995770193698826859762790479113062308975863428283798589097017957365590672835713863895712246676094993008985548024464030395443007480025079620363866193152298860635410053224484639158979864121027377255837396548653931285483865070903191974204864923589439190352993032676961005088404319792729916038927477474094858192679116146502863521484987023286193422239171712154568612530067201880859150042484947668670678405107153977068526645326383324039837473379697022624261377163163204493828299203980870340357510046733708501774838488222248753096417918793954837317546034884930540399950519191679471224053557093219350747155777569598163700852039470528193639241108443600686183546572496956218643721497262583322254659961604645585462993701658947042526444562415789958697265293564785696709268960442796501209877036845001246792761563917639959736383038665362727158

g^b mod p =

4116046620695933066832285256534418724107779992205720799935743972371563687620383783327424719396665449687938178193214952698336131699379861648113207956169499574005182063853102924755292845506262471329301240277031401312209687711427883948465928161110782751969552580451787052540164697735099369253619948958941630655511051619296131392197821987575429848264658934577688889155615145050480918561594129775760490735632255728098809700583965017196658531101013084326474277865655251213287725871678420376241901439097879386658420056919119973967264551107584485525537442884643379065403121253975718031032782719790076818413945341143157261205957499938963479817893107541948645774359056731729700335965844452066712238743995765602919548561681262366573815194145929420370183512324404671912281455859090458612780918001663308764073238447199488070126873048860279221761629281961046255219584327714817248626243962413613075956770018017385724999495117779149416882188

g^{ab} mod p =

3301669195241921493237617335984262446912241999588946540363315263943500990886273029798333395011830591981139878800667394199992313789707153070393178762584538767011245438495209794302333027775032650107245135512092795731832349343596366965069683257694895110289436988215186894965977582185407675178858364641602894716513645524907139614566085360133016497539758756106596557555674744381803579583602267087423481750455634370758409692308267670340611194376574669939893893482895996003389503722513369326735717434288230260146992320711161713922195996910968467141336433827457093761125005143009836512019611866134642676859265636245898172596372485581049036573719816844170539930826718273452528414333373254200883800592320891749460865366649848360413340316504386926391062876271575757583831289710534010374070317315095828076395094487046179839301350287596589383292751993079161318839043121329118930009948197899907586986108953591420279426874779423560221038468

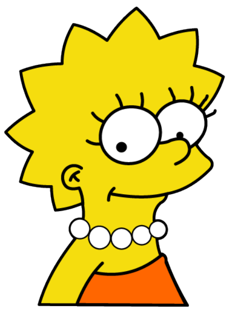
Example: KE using Elliptic Curves (128 bit security – p: 256 bit)

NIST Curve P-256

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

p = 115792089210356248762697446949407573530086143415290314195533631308867097853951

$$E(\mathbb{F}_p) : y^2 = x^3 - 3x + b$$



#E = 115792089210356248762697446949407573529996955224135760342422259061068512044369

P = (48439561293906451759052585252797914202762949526041747995844080717082404635286,
36134250956749795798585127919587881956611106672985015071877198253568414405109)

a=
8913064459124603357639770641462855023145028492835255603183721922317324614395

aP =
(84116208261315898167593067868200525612344221886333785331584793435449501658416,
102885655542185598026739250172885300109680266058548048621945393128043427650740)

bP =
(101228882920057626679704131545407930245895491542090988999577542687271695288383,
77887418190304022994116595034556257760807185615679689372138134363978498341594)

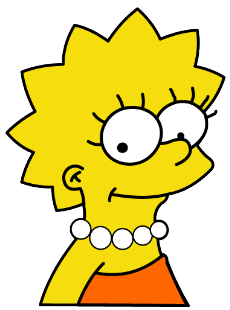
b=
10095557463932786418806938316190708032771910919058405391679781082193405190826

abP = (101228882920057626679704131545407930245895491542090988999577542687271695288383,
77887418190304022994116595034556257760807185615679689372138134363978498341594)

Diffie–Hellman(–Merkle) KE Protocol

- Now we are going to abstract away again the concrete setting and consider a group G of prime order q and generator g

$a \xleftarrow{\$} \mathbb{Z}_q; A \leftarrow g^a$ $\qquad$ $b \xleftarrow{\$} \mathbb{Z}_p; B \leftarrow g^b$



$\xrightarrow{\quad A \quad}$

$\xleftarrow{\quad B \quad}$

$K_A \leftarrow B^a$ $\qquad$ $K_B \leftarrow A^b$

Ok, how to prove security of this protocol?

- Under DL? Other means of computing shared key?
- Under CHD? Only the complete shared key protected?
- Under DDH?

* definitional framework and idea of formulating assumptions not known back in the 70ies

$\widehat{\mathrm{KE}}^{\mathrm{eav}}_{\mathcal{A},\Pi}$ Security

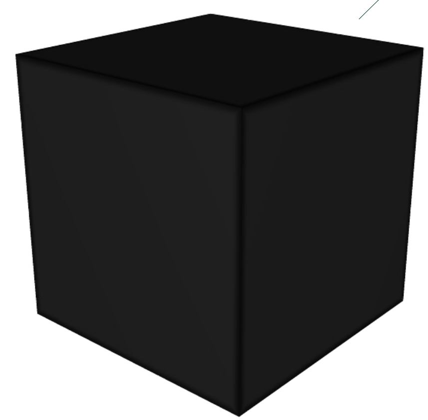
security parameter $n \in \mathbb{N}$ $(G, q, g) \xleftarrow{\$} \mathcal{G}(1^n)$

(G, q, g)

$\mathcal{A}$

(G, q, g)

Π



$(k^*, \text{ trans})$

b^*

$(k, \text{ trans})$

$b \xleftarrow{\$} \{0, 1\}$

if $b = 0$

$k^* := k$

else $k^* \xleftarrow{\$} \cancel{\{0, 1\}^n}$ G

A key-exchange protocol Π is secure in the presence of an eavesdropper if for every PPT adversary $\mathcal{A}$

$$Pr[b = b^*] \leq 1/2 + \mathrm{negl}(n)$$

Analysis of the DH(M) KE Protocol

THEOREM 10.3: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper (with respect to experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}$).

Proof: Let A be a PPT adversary.

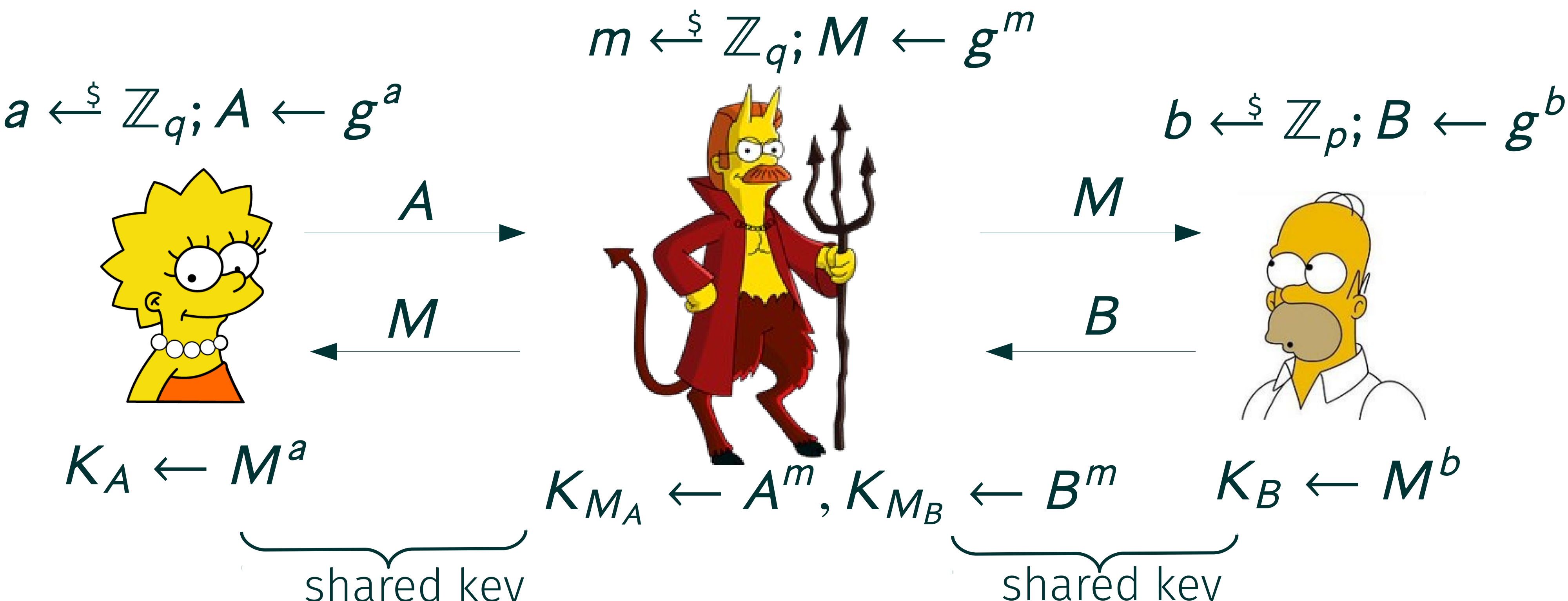
- Since Pr[b = 0] = Pr[b = 1] = ½, we have

$$
\begin{aligned}
&\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \\
&= 1/2 \cdot \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 | b = 0] + 1/2 \cdot \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 | b = 1] \\
&= 1/2 \cdot \Pr[\mathcal{A}(G, q, p, g^x, g^y, g^{xy}) = 0] + 1/2 \cdot \Pr[\mathcal{A}(G, q, p, g^x, g^y, g^z) = 1] \\
&= 1/2 \cdot (1 - \Pr[\mathcal{A}(G, q, p, g^x, g^y, g^{xy}) = 1]) + 1/2 \cdot \Pr[\mathcal{A}(G, q, p, g^x, g^y, g^z) = 1] \\
&= 1/2 + 1/2 \cdot (\Pr[\mathcal{A}(G, q, p, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(G, q, p, g^x, g^y, g^{xy}) = 1]) \\
&= 1/2 + 1/2 \cdot |\underbrace{\Pr[\mathcal{A}(G, q, p, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(G, q, p, g^x, g^y, g^{xy}) = 1}_{\leq \mathrm{negl}(n)}]| ,
\end{aligned}
$$

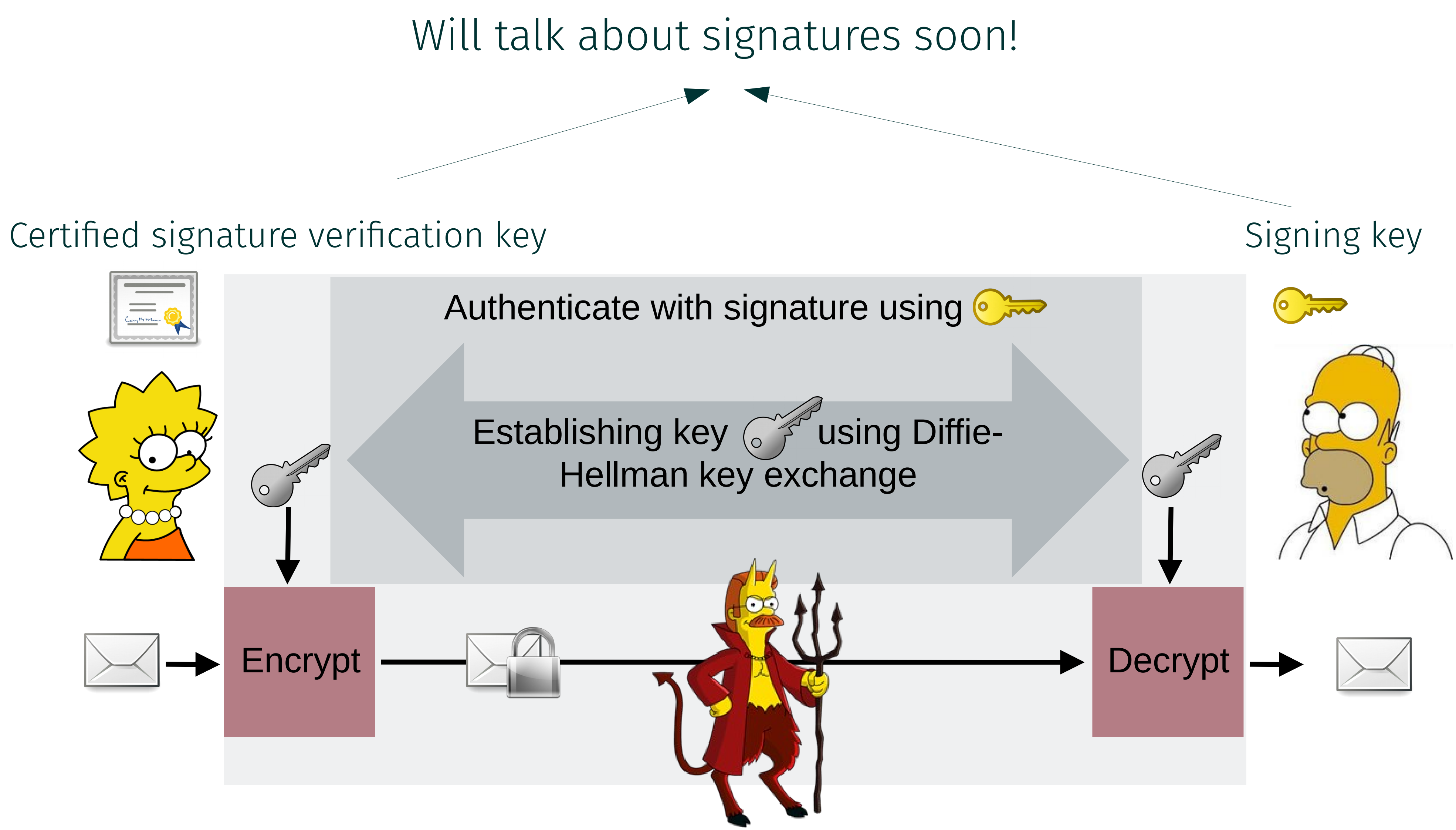
$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq 1/2 + 1/2 \cdot \mathrm{negl}(n).$$

Analysis of the DH(M) KE Protocol

- Summary
  - Can prove <u>eavesdropping security</u> under DDH (not surprising; the assumption was basically modeled to abstract the analysis of these protocols)
- What did we miss so far?
  - Active adversaries: Man-in-the-middle

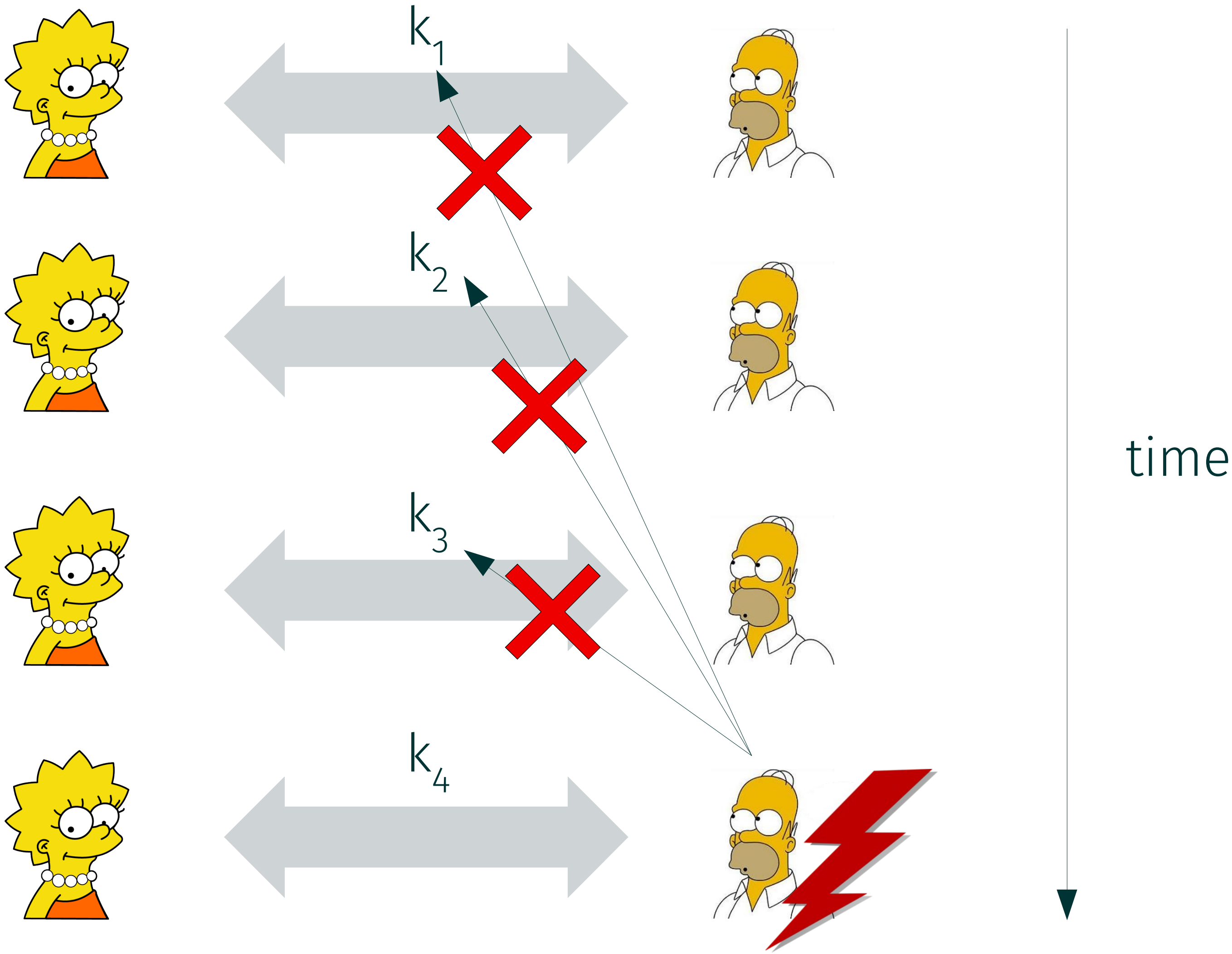


Countering man-in-the-middle attacks (Authenticated KE - AKE)



Perfect Forward Secrecy

Another important property: Perfect forward secrecy



Alternatives to DL based KE Protocols: Outlook

- Shor: computing discrete logarithms (and factoring) in polynomial time on a **quantum computer**
  - If we have a sufficiently powerful quantum computer, then DL and ECDL (as well as factoring) based systems will be dead

Peter Shor

- What to do if this should happen?
  - Post-quantum cryptography: (asymmetric) cryptography that is conjectured to resists attacks using classical and quantum computers
- Very active field of research
  - Lattices
  - Codes
  - Isogenies (e.g., on supersingular elliptic curves – weak for EC crypto but good for PQ)
  - Etc.

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

https://csrc.nist.gov/projects/post-quantum-cryptography