In the beginning, there was *symmetric* encryption.

If you had the key you could encrypt ...

If you had the key you could encrypt ...

Message:	**ATTACK AT DAWN**
Key: +3	↓↓↓↓↓↓ ↓↓ ↓↓↓↓
Ciphertext:	**DWWDFN DW GDZQ**

If you had the key you could decrypt …

If you had the key you could decrypt …

Message:	**ATTACK AT DAWN**
Key: +3	↑↑↑↑↑↑ ↑↑ ↑↑↑↑
Ciphertext:	**DWWDFN DW GDZQ**

… and some people were happy.

Then, there was asymmetric encryption.

Some people encrypted …

Some people encrypted …

… others decrypted.

E-commerce ensued …

E-commerce ensued …

… and more people were happy.

The first and most used asymmetric cipher was RSA.

The first and most used asymmetric cipher was RSA.

$$E(m) = m^e \pmod{n}$$

Some people noticed the algebraic structure …

Some people noticed the algebraic structure …

$$E(m_1) = {m_1}^e \qquad E(m_2) = {m_2}^e$$

Some people noticed the algebraic structure …

$$E(m_1) = {m_1}^e \qquad E(m_2) = {m_2}^e$$

Ergo …

Some people noticed the algebraic structure …

$$E(m_1) = {m_1}^e \qquad E(m_2) = {m_2}^e$$

Ergo …

$$E(m_1) \times E(m_2)$$

Some people noticed the algebraic structure …

$$E(m_1) = m_1{}^e \qquad E(m_2) = m_2{}^e$$

Ergo …

$$E(m_1) \times E(m_2)$$

$$= m_1{}^e \times m_2{}^e$$

Some people noticed the algebraic structure …

$$E(m_1) = {m_1}^e \qquad E(m_2) = {m_2}^e$$

Ergo …

$$\begin{aligned} E(m_1) \times E(m_2) \\ &= {m_1}^e \times {m_2}^e \\ &= (m_1 \times m_2)^e \end{aligned}$$

Some people noticed the algebraic structure …

$$E(m_1) = m_1{}^e \qquad E(m_2) = m_2{}^e$$

Ergo …

$$\begin{aligned} & E(m_1) \times E(m_2) \\ & \qquad = m_1{}^e \times m_2{}^e \\ & \qquad = (m_1 \times m_2)^e \\ & \qquad = E(m_1 \times m_2) \end{aligned}$$

They looked for interesting applications …

They looked for interesting applications …

… and they failed.

People mused …

People mused …

… if only RSA worked additively …

People mused …

… if only RSA worked additively …

we could compute sums …

People mused …

… if only RSA worked additively …

we could compute sums …

and averages …

People mused …

… if only RSA worked additively …

we could compute sums …

and averages …

and tally elections …

I was one of those musing.

I was one of those musing.

An additive encryption homomorphism …

I was one of those musing.

An additive encryption homomorphism …

$$E(m, r) = r^e c^m$$

$$E(m_1, r_1) = {r_1}^e c^{m_1} \qquad E(m_2, r_2) = {r_2}^e c^{m_2}$$

$$E(m_1, r_1) = r_1{}^e c^{m_1} \qquad E(m_2, r_2) = r_2{}^e c^{m_2}$$

$$E(m_1, r_1) \times E(m_2, r_2)$$

$$E(m_1, r_1) = \ {r_1}^e c^{m_1} \qquad E(m_2, r_2) = \ {r_2}^e c^{m_2}$$

$$E(m_1, r_1) \times E(m_2, r_2)$$
$$= {r_1}^e c^{m_1} \times {r_2}^e c^{m_2}$$

$$E(m_1, r_1) = r_1{}^e c^{m_1} \qquad E(m_2, r_2) = r_2{}^e c^{m_2}$$

$$\begin{aligned} E(m_1, r_1) \times E(m_2, r_2) \\ &= r_1{}^e c^{m_1} \times r_2{}^e c^{m_2} \\ &= (r_1 r_2)^e c^{m_1+m_2} \end{aligned}$$

$$E(m_1, r_1) = \ {r_1}^e c^{m_1} \qquad E(m_2, r_2) = \ {r_2}^e c^{m_2}$$

$$\begin{aligned} &E(m_1, r_1) \times E(m_2, r_2) \\ &\quad = {r_1}^e c^{m_1} \times {r_2}^e c^{m_2} \\ &\quad = (r_1 r_2)^e c^{m_1 + m_2} \\ &\quad = E(m_1 + m_2, r_1 r_2) \end{aligned}$$

$$E(m_1, r_1) = r_1{}^e c^{m_1} \qquad E(m_2, r_2) = r_2{}^e c^{m_2}$$

$$\begin{aligned} E(m_1, r_1) \times E(m_2, r_2) \\ &= r_1{}^e c^{m_1} \times r_2{}^e c^{m_2} \\ &= (r_1 r_2)^e c^{m_1 + m_2} \\ &= E(m_1 + m_2, r_1 r_2) \end{aligned}$$

The product of encryptions of two messages is *an* encryption of the sum of the two messages.

I used this to build verifiable election systems …

I used this to build verifiable election systems …

… and I was *really* happy …

I used this to build verifiable election systems …

… and I was *really* happy …

and few others cared.

What people really wanted was the ability to do arbitrary computing on encrypted data…

What people really wanted was the ability to do arbitrary computing on encrypted data…

… and this required the ability to compute *both* sums *and* products …

What people really wanted was the ability to do arbitrary computing on encrypted data…

… and this required the ability to compute *both* sums *and* products …

… on the same data set!

People tried to do this for years …

People tried to do this for years …

… and years …

People tried to do this for years …

… and years …

… and years …

People tried to do this for years …

… and years …

… and years …

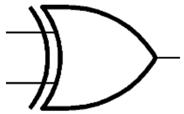
… with no success.


STAR WARS
RETURN OF THE JEDI


STAR WARS
THE CLONE WARS
IN THEATERS AUGUST 15

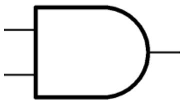
WHY does ADD AND MULTIPLY help?

WHY does ADD AND MULTIPLY help?



XOR (add mod 2)

0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0



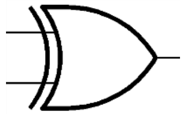
AND (mult mod 2)

0 AND 0	0
1 AND 0	0
0 AND 1	0
1 AND 1	1

WHY does ADD AND MULTIPLY help?

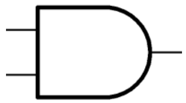
... because {XOR,AND} is Turing-complete ...

(any function can be written as a combination of XOR and AND gates)



XOR (add mod 2)

0 XOR 0	0
1 XOR 0	1
0 XOR 1	1
1 XOR 1	0



AND (mult mod 2)

0 AND 0	0
1 AND 0	0
0 AND 1	0
1 AND 1	1

WHY does ADD AND MULTIPLY help?

... because {XOR,AND} is Turing-complete ...

(any function can be written as a combination of XOR and AND gates)

Example: Searching a database

DB

0
1
1
0

index

$I = i_1 i_0$

$\longleftarrow$

answer= DB_I

$\longrightarrow$

WHY does ADD AND MULTIPLY help?

… because {XOR,AND} is Turing-complete …

(any function can be written as a combination of XOR and AND gates)

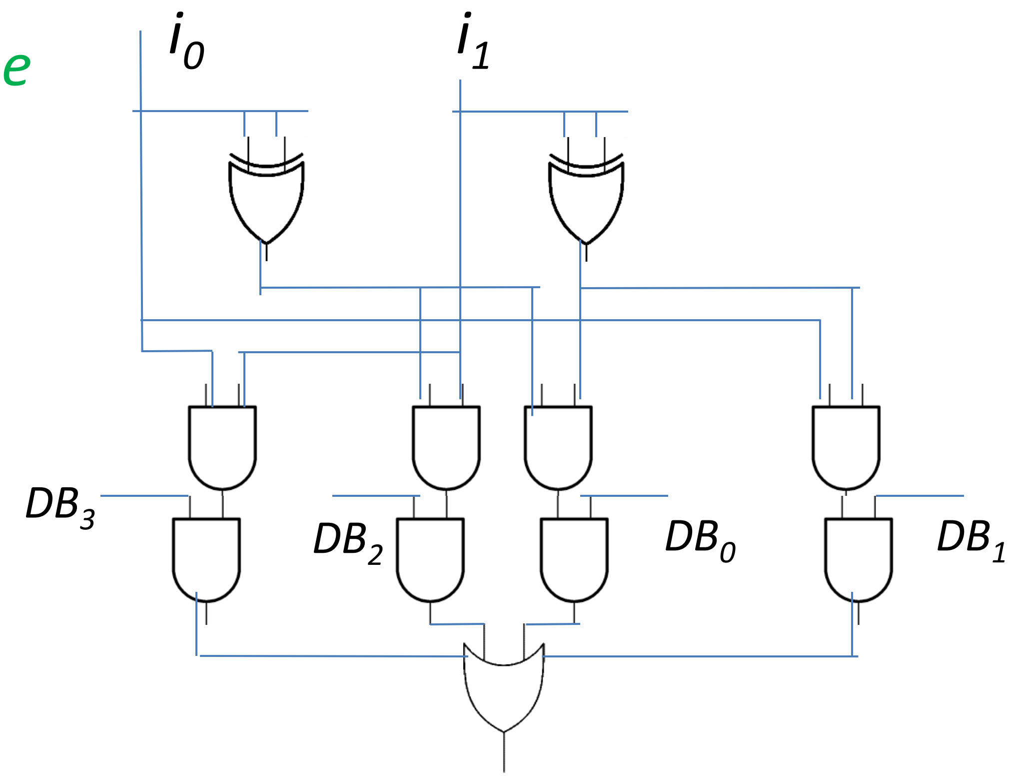
Example: Searching a database

DB
0
1
1
0

index

$I = i_1 i_0$

answer= DB_I

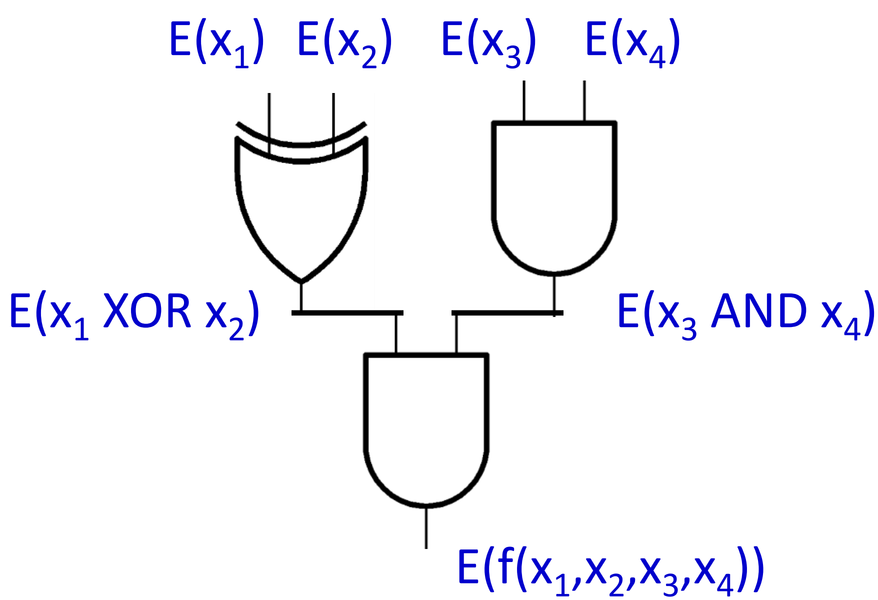


WHY does ADD AND MULTIPLY help?

… because {XOR,AND} is Turing-complete …

… if you can compute XOR and AND on encrypted bits…

… you can compute ***ANY*** function on encrypted inputs…



This is A M A Z I N G!

Private bing Search

Private Cloud computing

This is A M A Z I N G!

Private bing Search

Private Cloud computing

In general,

Delegate *processing* of data

without giving away *access* to it

People tried to compute both AND and XOR on encrypted bits …

… for years …

… and years …

… with no success.

Well, actually, there were some ***partial*** answers …

Fully homomorphic

Josh's system

MANY add

ZERO mult

MANY add

MANY mult

Well, actually, there were some ***partial*** answers …

Fully homomorphic

Josh's system

Boneh,Goh & Nissim

MANY add
ZERO mult

MANY add
1 mult

MANY add
MANY mult

… and some bold attempts [Fellows-Koblitz] …

… which were quickly broken …

… until, in October 2008 …

… until, in October 2008 …

… *Craig Gentry* came up with the first

fully homomorphic encryption scheme …

How does it work?

What is the magic?

Gentry’s scheme was complex …

… it used advanced algebraic number theory …

Some of us asked: can we make this really simple? …

Some of us asked: can we make this really simple? …

Polynomials?

$(x^2 + 6x + 1)$ + $(x^2 - 6x) = (2x^2 + 1)$

$(x^2 + 6x + 1)$ X $(x^2 - 6x) = (x^4 - 35x^2 - 6x)$

Some of us asked: can we make this really simple? …

Polynomials?

$$(x^2 + 6x + 1) + (x^2 - 6x) = (2x^2 + 1)$$

$$(x^2 + 6x + 1) \text{ X } (x^2 - 6x) = (x^4 - 35x^2 - 6x)$$

Matrices?

$$\begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix} + \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix} X \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ -1 & 3 \end{pmatrix}$$

Some of us asked: can we make this really simple? …

Polynomials?

$$(x^2+6x+1)\ +(x^2-6x)=(2x^2+1)$$

$$(x^2+6x+1)\ \text{X}\ (x^2-6x)=(x^4-35x^2-6x)$$

Matrices?

$$\begin{pmatrix}1 & 0\\ 1 & 2\end{pmatrix}+\begin{pmatrix}-1 & 1\\ 0 & 1\end{pmatrix}=\begin{pmatrix}0 & 1\\ 1 & 3\end{pmatrix}$$

$$\begin{pmatrix}1 & 0\\ 1 & 2\end{pmatrix}X\begin{pmatrix}-1 & 1\\ 0 & 1\end{pmatrix}=\begin{pmatrix}-1 & 1\\ -1 & 3\end{pmatrix}$$

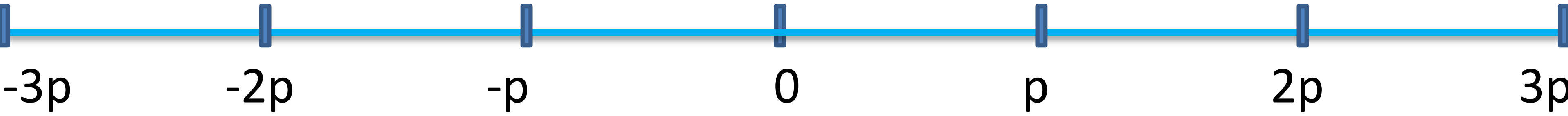
How about ***integers?!?*** [Gentry, Halevi, van Dijk, **V.**]

$$2\ +\ 3=5$$

$$2\ \text{X}\ 3=6$$

TODAY: Secret-key (Symmetric-key) Encryption

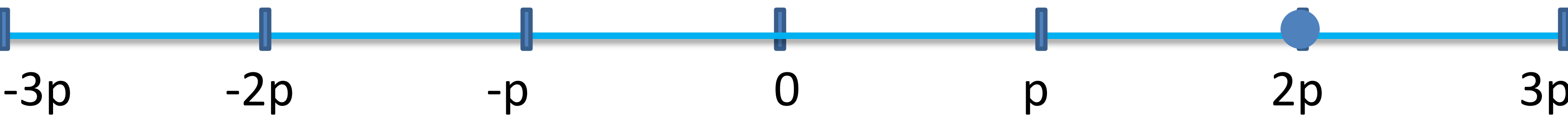
Secret key: large *odd* number **p**



Secret key: large *odd* number **p**

*To Encrypt a bit **b**:*

– pick a (random) “large” multiple of p, say **q·p**



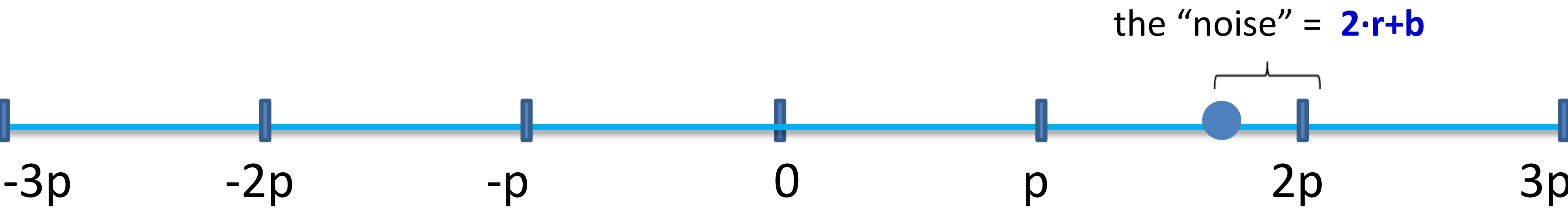
Secret key: large *odd* number **p**

*To Encrypt a bit **b**:*

– pick a (random) “large” multiple of p, say **q·p**

– pick a (random) “small” number **2·r+b**

(this is even if b=0, and odd if b=1)

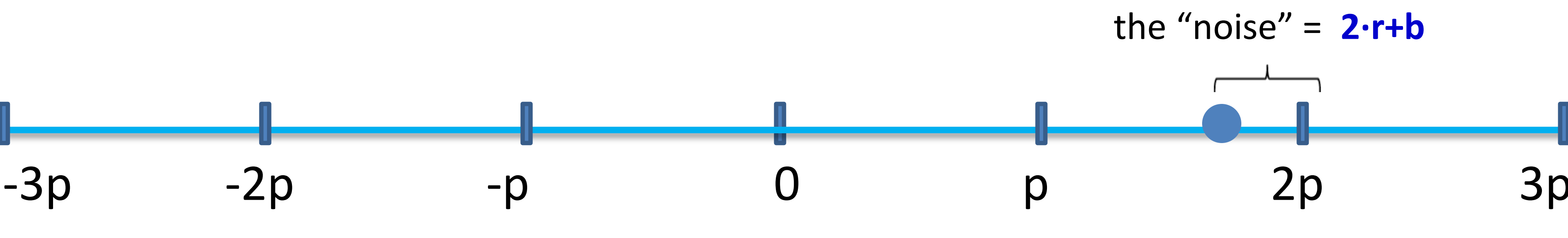


Secret key: large *odd* number **p**

*To Encrypt a bit **b**:*

- pick a (random) “large” multiple of p, say **q·p**
- pick a (random) “small” number **2·r+b**

  (this is even if b=0, and odd if b=1)
- Ciphertext **c = q·p+2·r+b**



Secret key: large *odd* number **p**

*To Encrypt a bit **b**:*

– pick a (random) “large” multiple of p, say **q·p**

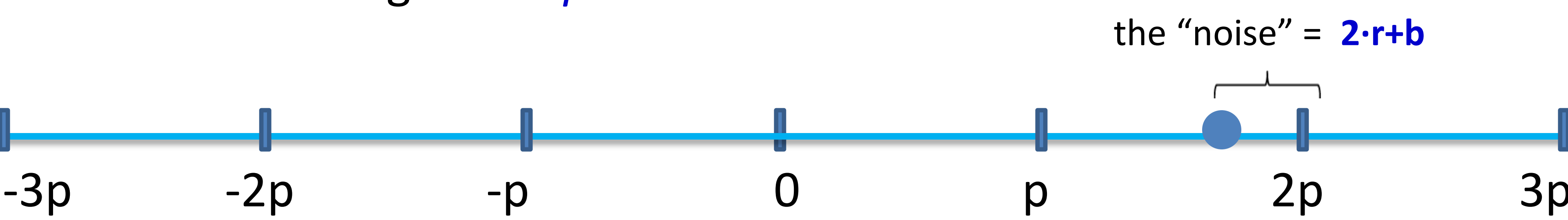
– pick a (random) “small” number **2·r+b**

(this is even if b=0, and odd if b=1)

– Ciphertext **c = q·p+2·r+b**

*To Decrypt a ciphertext **c**:*

Taking ***c*** *mod p* recovers the noise



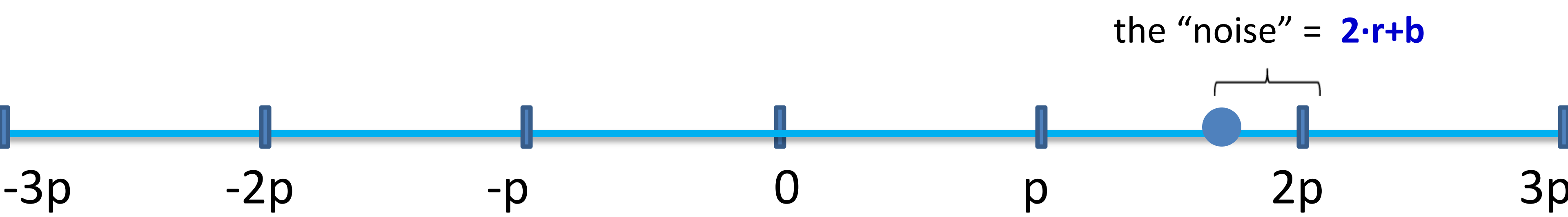
How secure is this?

... if there were no noise (think $r=0$)

... and I give you two encryptions of 0 (q_1p & q_2p)

... then you can recover the secret key p

$$= \mathrm{GCD}(q_1p,\ q_2p)$$



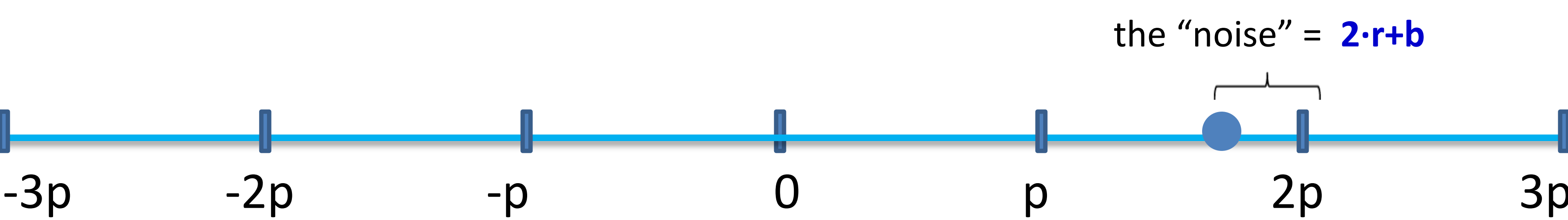
How secure is this?

… but if there is noise

… the GCD attack doesn't work

… and neither does any attack (we believe)

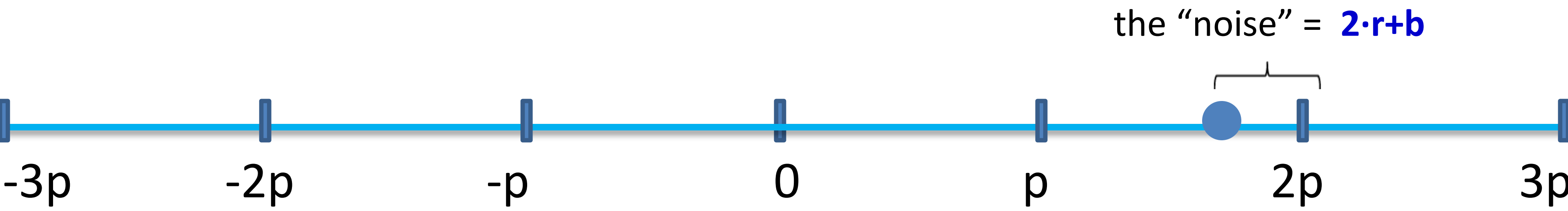
… this is called the *approximate GCD assumption*



XORing two encrypted bits:

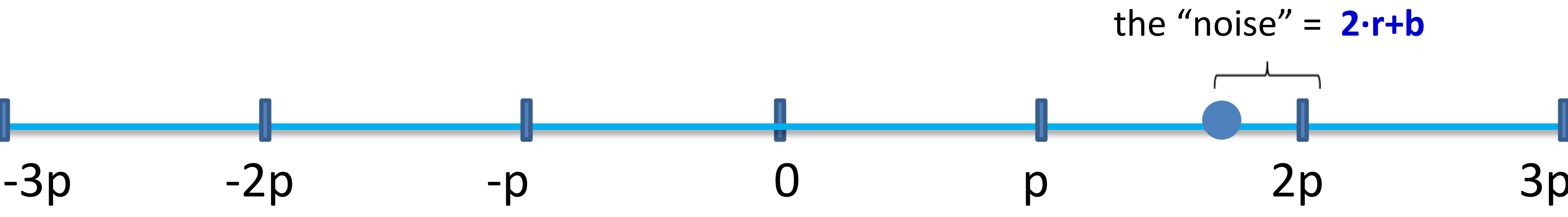
- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$

- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$



XORing two encrypted bits:

- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$
- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$
- $\mathbf{c_1 + c_2} = \mathbf{p \cdot}(q_1 + q_2) + \mathbf{2 \cdot}(r_1 + r_2) + (b_1 + b_2)$

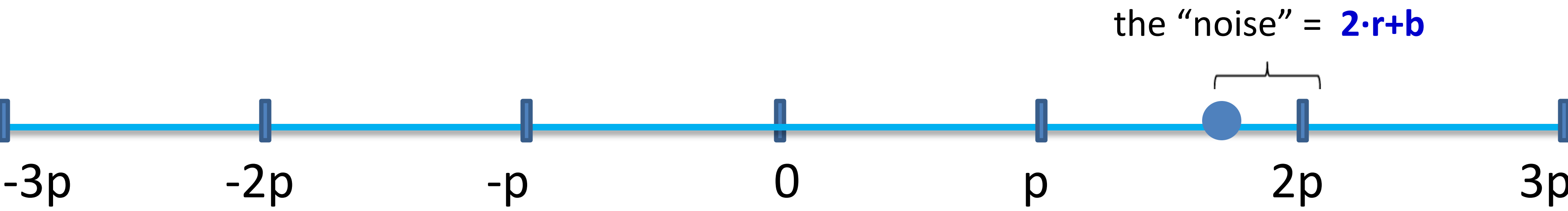


XORing two encrypted bits:

- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$
- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$
- $\mathbf{c_1 + c_2} = \mathbf{p\cdot}(q_1 + q_2) + \mathbf{2\cdot}(r_1 + r_2) + (b_1 + b_2)$

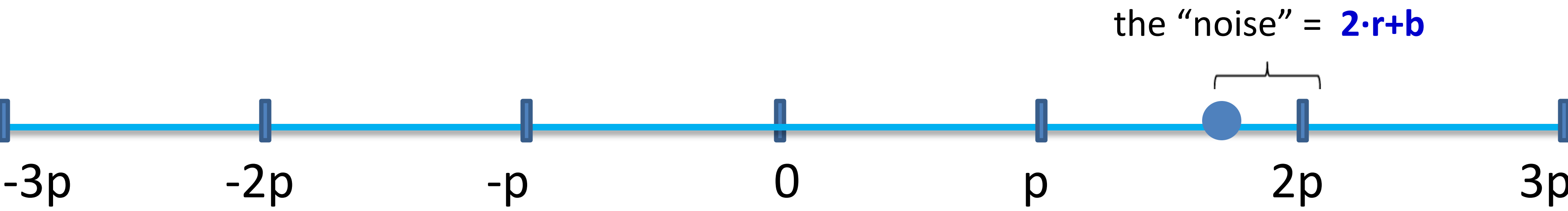
Odd if $b_1=0, b_2=1$ (or)
$b_1=1, b_2=0$

Even if $b_1=0, b_2=0$ (or)
$b_1=1, b_2=1$



XORing two encrypted bits:

- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$
- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$
- $\mathbf{c_1 + c_2} = \mathbf{p\cdot}(q_1 + q_2) + \underbrace{\boxed{\mathbf{2\cdot}(r_1+r_2) + (b_1+b_2)}}_{\textbf{\textit{lsb}}= \boldsymbol{b_1} \textbf{\textit{ XOR }} \boldsymbol{b_2}}$

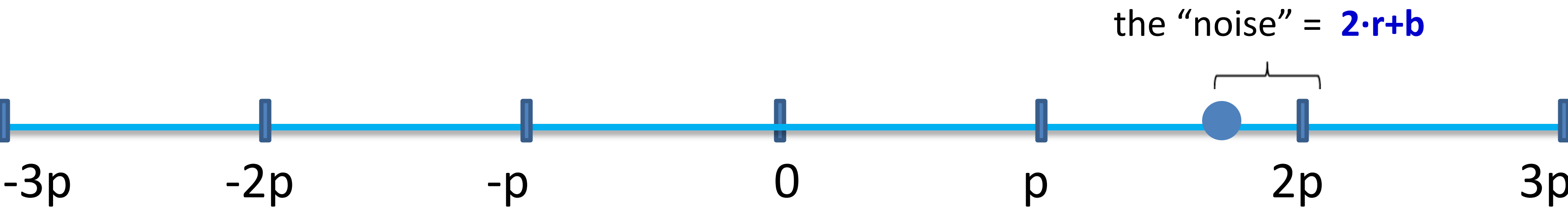


ANDing two encrypted bits:

- $\mathbf{c_1} = q_1 \cdot p + (2 \cdot r_1 + b_1)$

- $\mathbf{c_2} = q_2 \cdot p + (2 \cdot r_2 + b_2)$

- $\mathbf{c_1 c_2} = \mathbf{p\cdot}(c_2 \cdot q_1 + c_1 \cdot q_2 - q_1 \cdot q_2) + \mathbf{2\cdot}(r_1 r_2 + r_1 b_2 + r_2 b_1) + b_1 b_2$

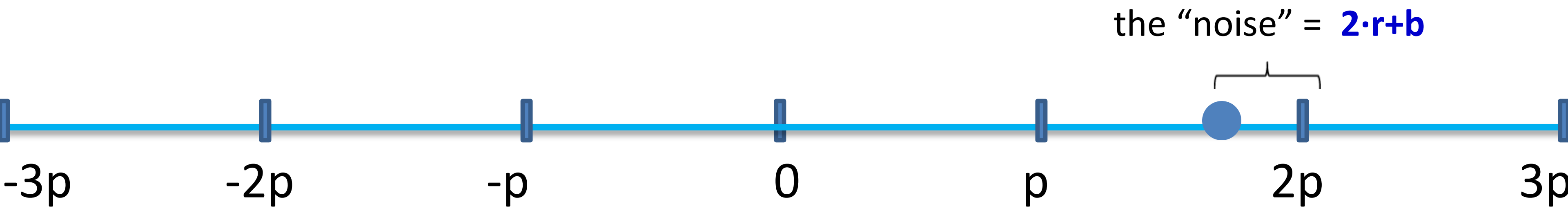


ANDing two encrypted bits:

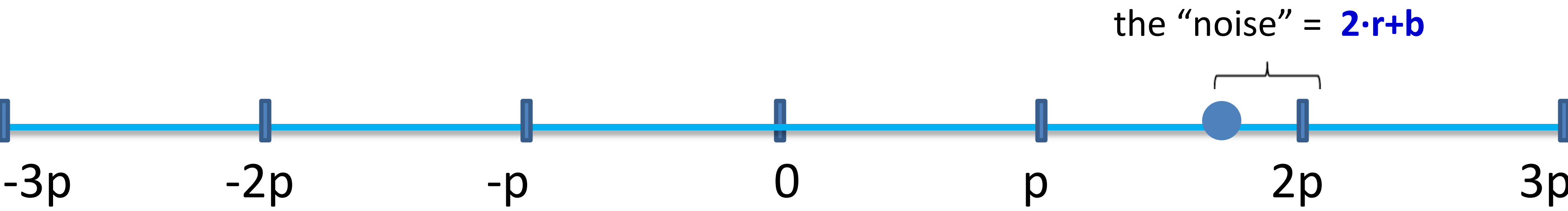
- $\mathbf{c_1} = q_1\cdot p + (2\cdot r_1 + b_1)$

- $\mathbf{c_2} = q_2\cdot p + (2\cdot r_2 + b_2)$

- $\mathbf{c_1 c_2} = \mathbf{p\cdot}(c_2\cdot q_1+c_1\cdot q_2-q_1\cdot q_2) + \underbrace{\mathbf{2\cdot}(r_1r_2+r_1b_2+r_2b_1) + b_1b_2}_{\textit{lsb}= b_1 \text{ AND } b_2}$

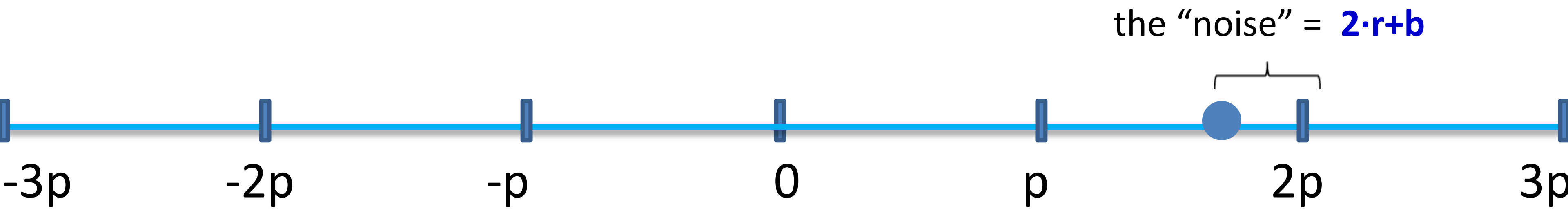


the noise grows!



the noise grows!

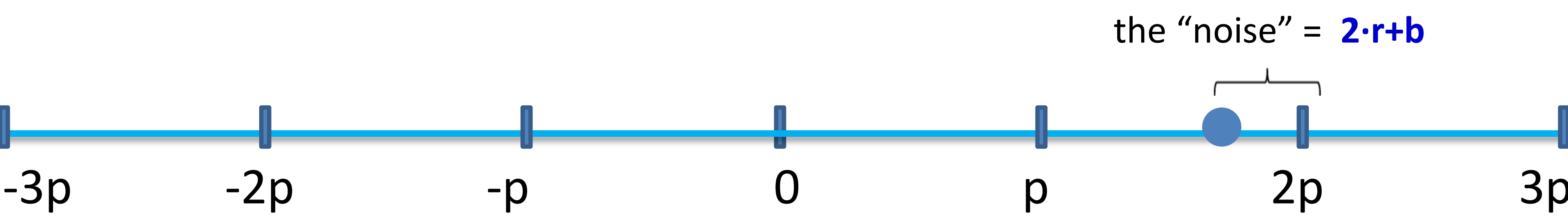
- $c_1+c_2 = p\cdot(q_1+q_2) + \underbrace{2\cdot(r_1+r_2) + (b_1+b_2)}_{\textit{noise= 2 * (initial noise)}}$



the noise grows!

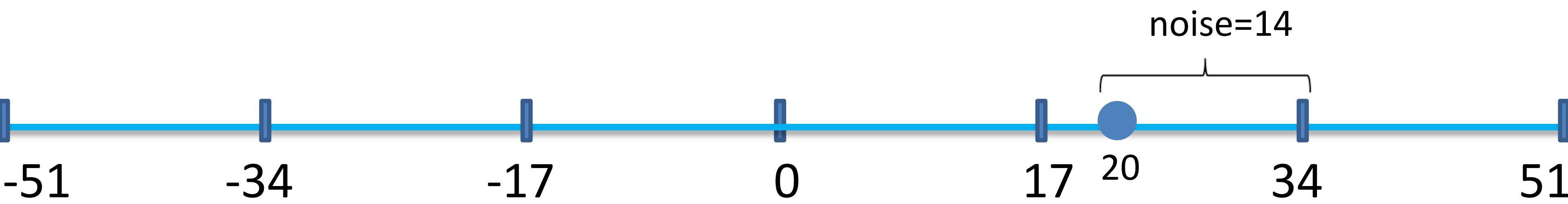
– $\mathbf{c_1+c_2} = \mathbf{p}\cdot(q_1 + q_2) + \underbrace{\mathbf{2}\cdot(r_1+r_2) + (b_1+b_2)}_{\textit{noise} = 2 * (\textit{initial noise})}$

– $\mathbf{c_1 c_2} = \mathbf{p}\cdot(c_2\cdot q_1+c_1\cdot q_2-q_1\cdot q_2) + \underbrace{\mathbf{2}\cdot(r_1 r_2+r_1 b_2+r_2 b_1) + b_1 b_2}_{\textit{noise} = (\textit{initial noise})^2}$

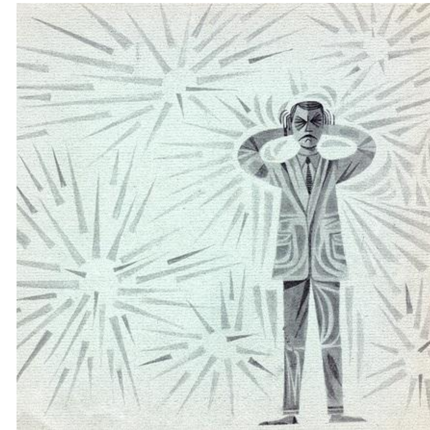


the noise grows!

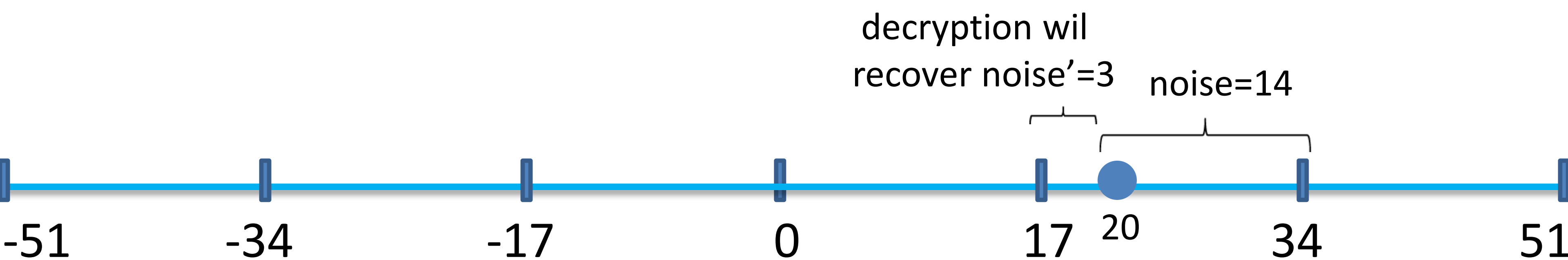
… so what’s the problem?



the noise grows!



… so what's the problem?

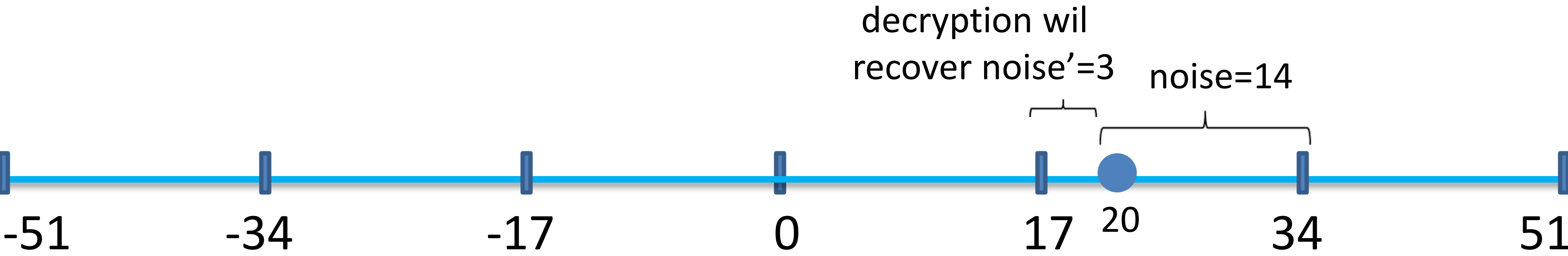


the noise grows!

… so what’s the problem?

If the |noise| > p/2, then …

decryption will output an incorrect bit



So, what did we accomplish?

… we can do lots of additions and

… some multiplications

(= a “somewhat homomorphic” encryption)

So, what did we accomplish?

… we can do lots of additions and

… some multiplications

(= a "somewhat homomorphic" encryption)

… enough to do many useful tasks, e.g., database search, spam filtering etc.

So, what did we accomplish?

… we can do lots of additions and

… some multiplications

(= a "somewhat homomorphic" encryption)

… enough to do many useful tasks, e.g.,
database search, spam filtering etc.

But I promised much more …

Josh's system
Boneh,Goh & Nissim
Fully homomorphic
MANY add
ZERO mult
MANY add
1 mult
WE ARE HERE!
MANY add
MANY mult

Gentry's ***"bootstrapping theorem" …***

Gentry's ***"bootstrapping theorem"*** ***…***

… If you can go a (large) part of the way,

then you can go all the way.

Gentry's ***"bootstrapping theorem"*** ...

... If you can go a (large) part of the way,

then you can go all the way.

[HOW? WE'LL SEE IN A BIT]

How efficient is all this?

… can I buy a homomorphic encryption software and start encrypting my data?

How efficient is all this?

… can I buy a homomorphic encryption software and start encrypting my data?

… well, not quite yet

How efficient is all this?

… can I buy a homomorphic encryption software and start encrypting my data?

… well, not quite yet

… encrypting a bit takes ~19s (!) with the current best implementation

How efficient is all this?

… can I buy a homomorphic encryption software and start encrypting my data?

… well, not quite yet

… encrypting a bit takes ~19s (!) with the current best implementation

… it takes 99 min to encrypt this sentence

How efficient is all this?

… can I buy a homomorphic encryption software and start encrypting my data?

… well, not quite yet

… encrypting a bit takes ~19s (!) with the current best implementation

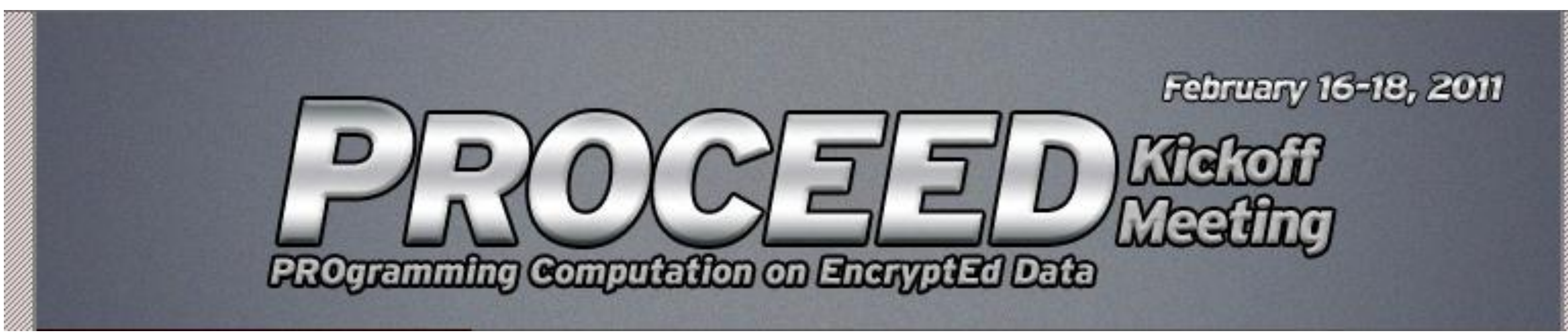
… but we are improving rapidly…

… a number of new, more efficient schemes

… optimized implementation efforts

(in hardware and software)

… and a $20M DARPA project to fund all this



… a number of new, more efficient schemes

… optimized implementation efforts

(in hardware and software)

… and a $20M DARPA project to fund all this

So, watch out for new developments!

References:

[1] "Computing arbitrary functions of Encrypted Data", Craig Gentry, Communications of the ACM 53(3), 2010.

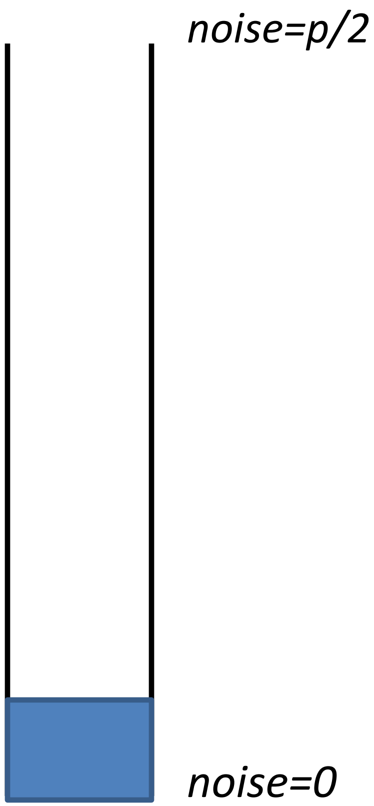
[2] "Fully Homomorphic Encryption from the Integers",
Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan
http://eprint.iacr.org/2009/616, Eurocrypt 2010.

[3] "Implementing Gentry's Fully Homomorphic Encryption",
Craig Gentry and Shai Halevi
https://researcher.ibm.com/researcher/files/us-shaih/fhe-implementation.pdf, Eurocrypt 2011.

Gentry's ***"bootstrapping method" …***

… If you can go a (large) part of the way,

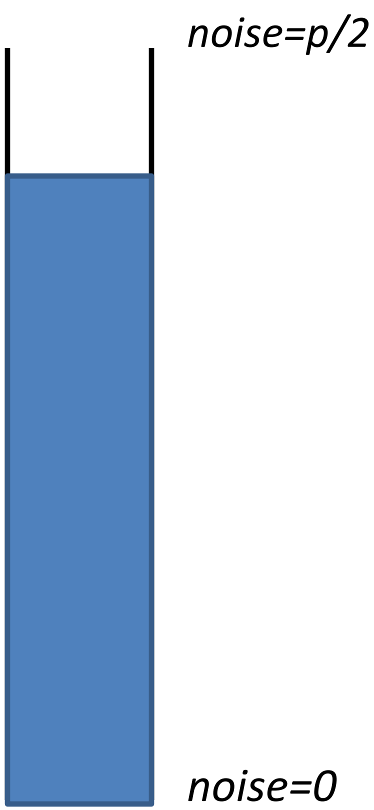
then you can go all the way…



*Gentry's **"bootstrapping method" …***

… If you can go a (large) part of the way,

then you can go all the way…



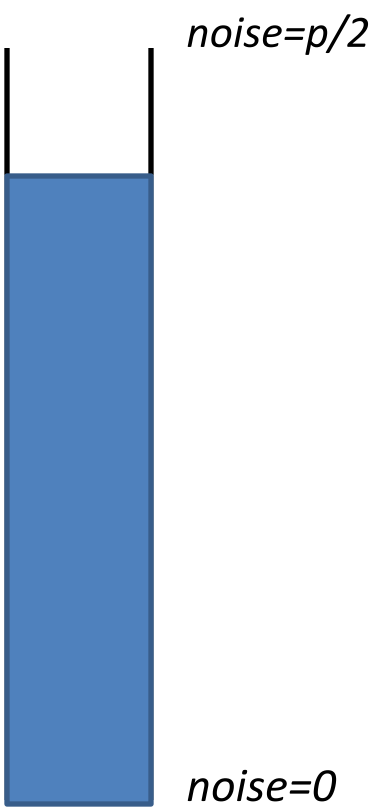
Problem: Add and Mult increase noise

(Add doubles, Mult squares the noise)

Gentry's ***"bootstrapping method" …***

… If you can go a (large) part of the way,

then you can go all the way…



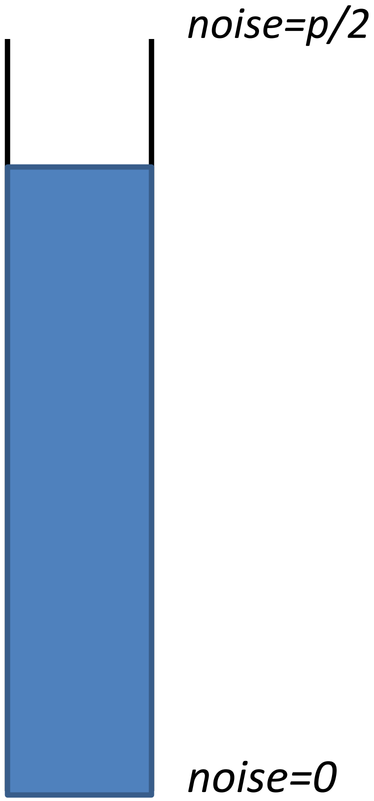
Problem: Add and Mult increase noise

(Add doubles, Mult squares the noise)

So, we want to do *noise-reduction*

Let's think…

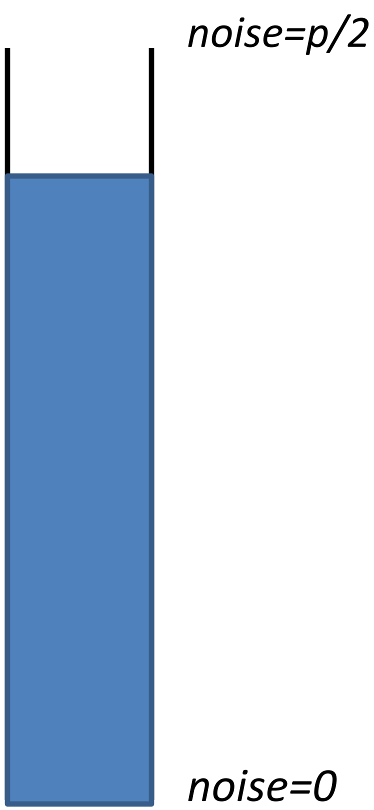
… What is the best noise-reduction procedure?



Let's think…

… What is the best noise-reduction procedure?

… something that kills all noise



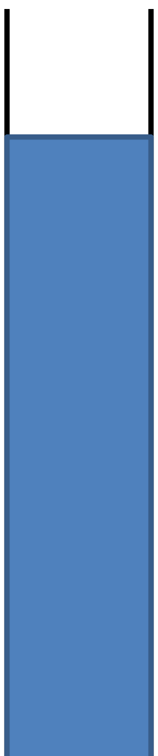
Let's think…

… What is the best noise-reduction procedure?

… something that kills all noise

noise=p/2

… and recovers the message



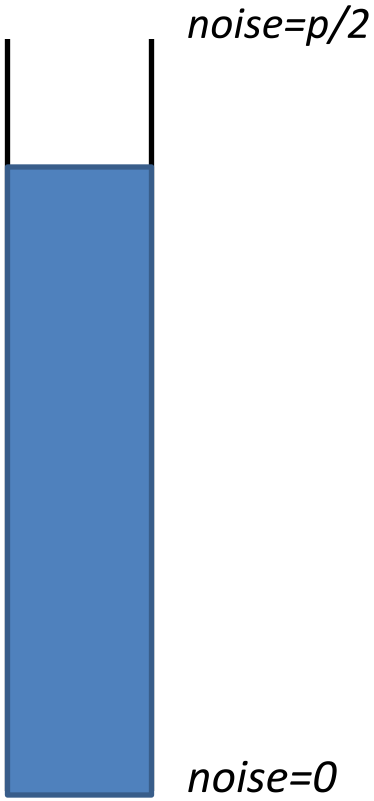
noise=0

Let's think…

… What is the best noise-reduction procedure?

… something that kills all noise

… and recovers the message



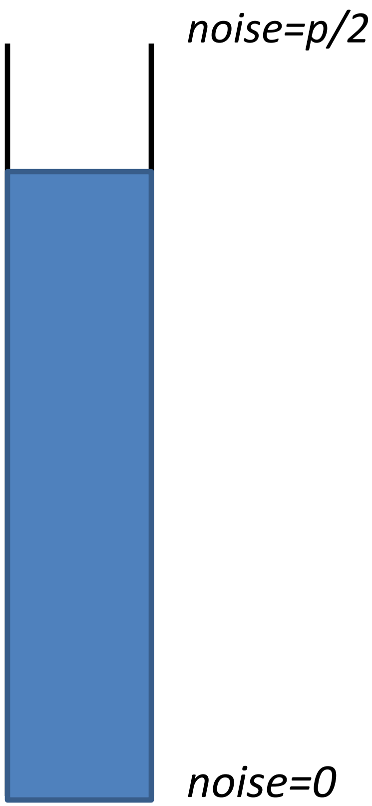
Decryption!

Let's think…

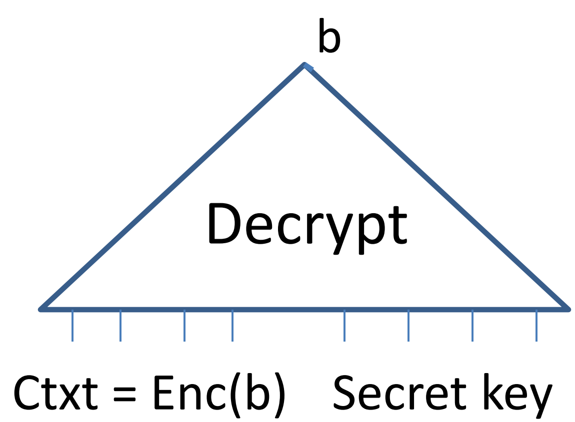
… What is the best noise-reduction procedure?

… something that kills all noise

… and recovers the message



Decryption!

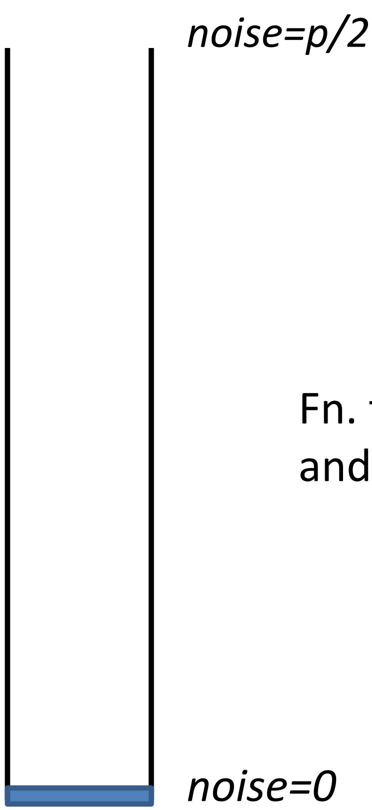


Let's think…

… What is the best noise-reduction procedure?

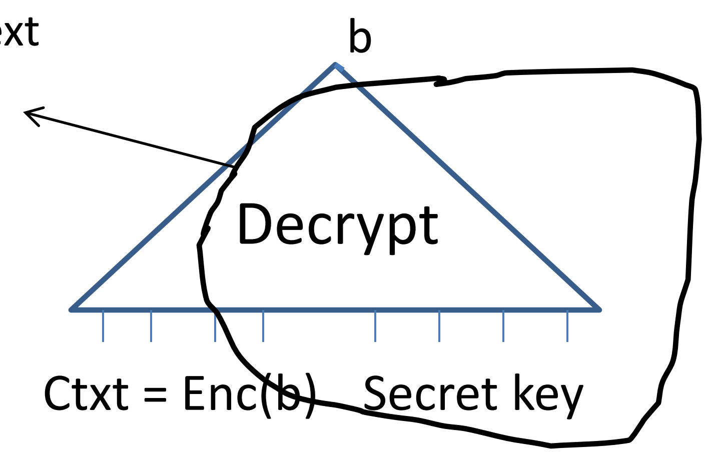
… something that kills all noise

… and recovers the message



Decryption!

Fn. that acts on ciphertext
and eliminates noise



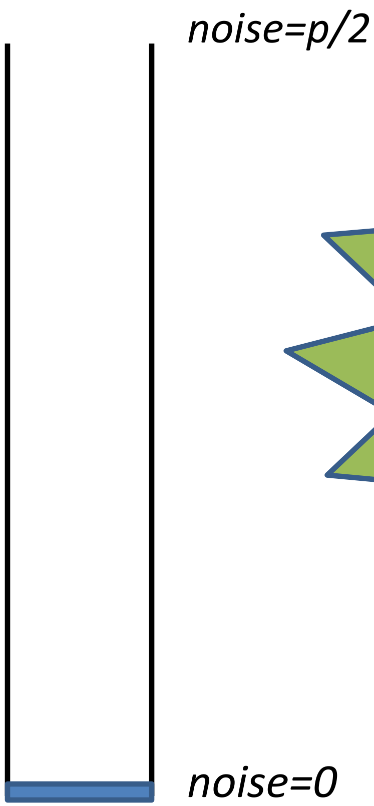
Let’s think…

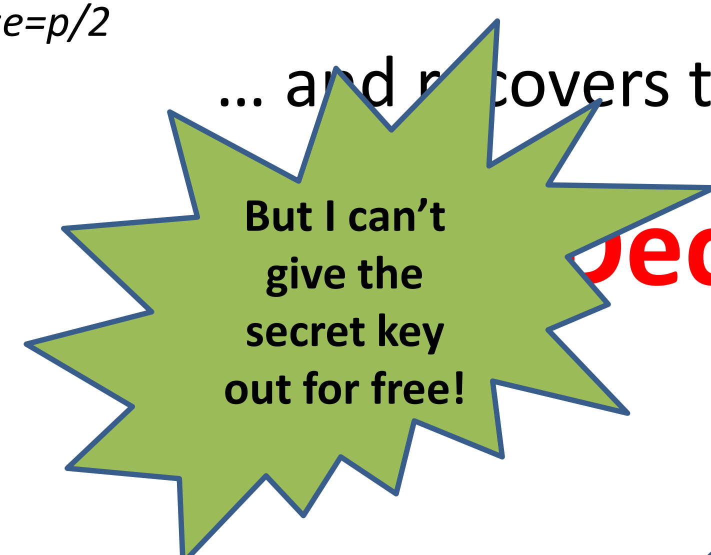
… What is the best noise-reduction procedure?

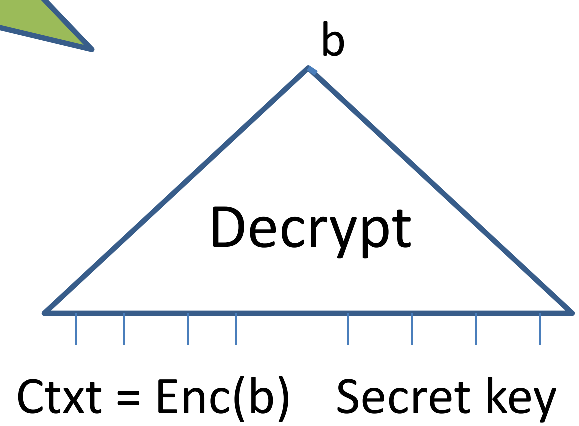
… something that kills all noise

… and recovers the message

Decryption!

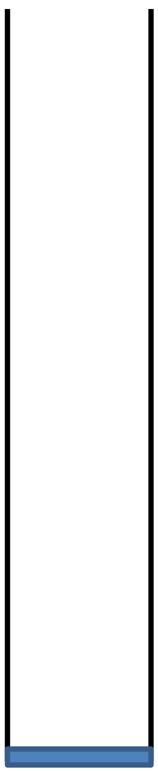




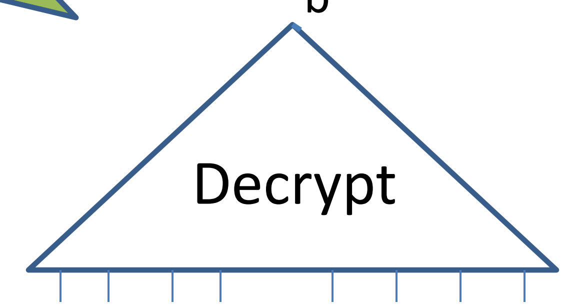


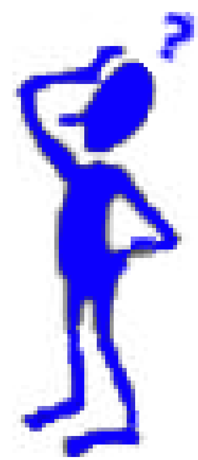
Let's think…

… I want to reduce noise *without letting you decrypt*



But I can't give the secret key out for free!

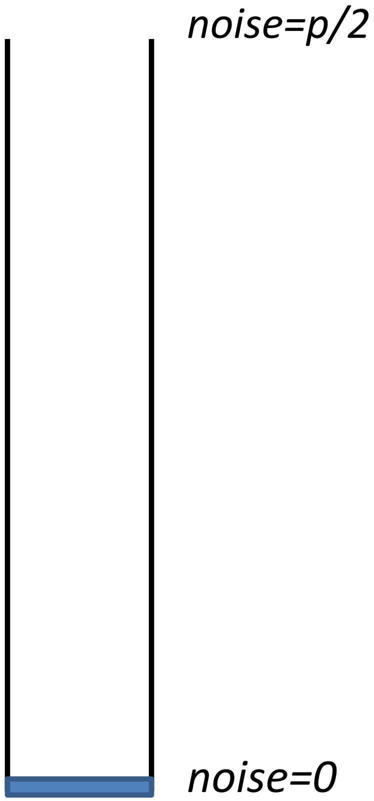


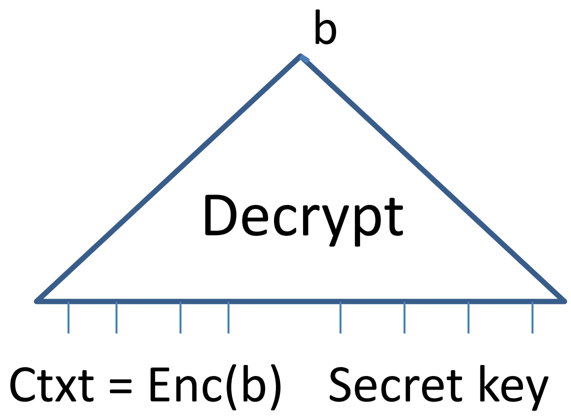


KEY IDEA:

… I cannot release the secret key (lest everyone sees my data)

… but *I can release Enc(secret key)*



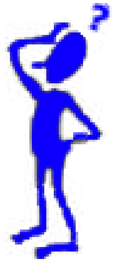


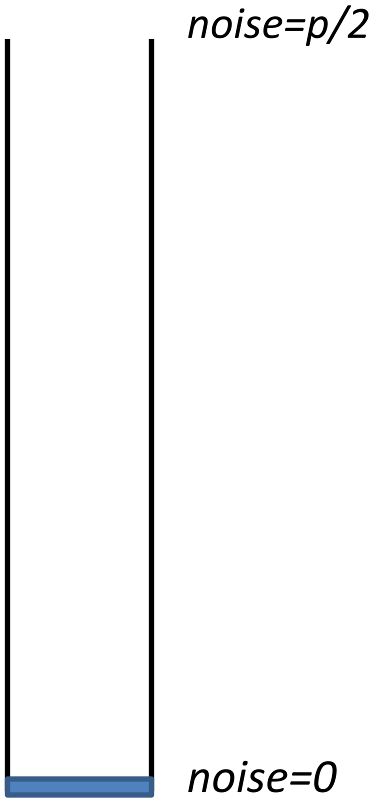
KEY IDEA:

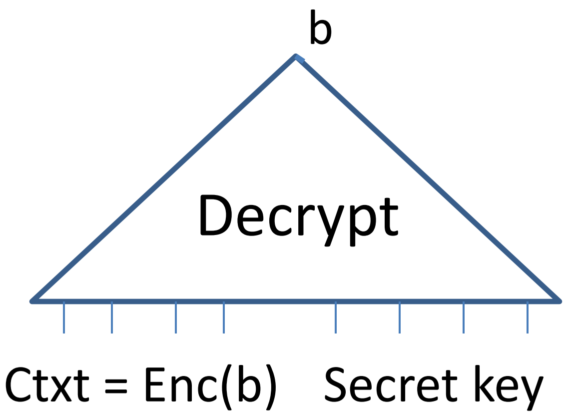
… I cannot release the secret key (lest everyone sees my data)

… but *I can release Enc(secret key)*

… called “Circular Encryption”



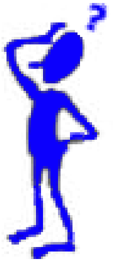




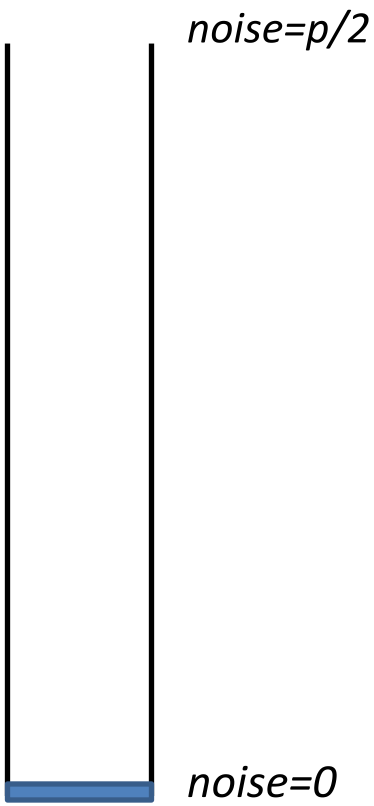
KEY IDEA:

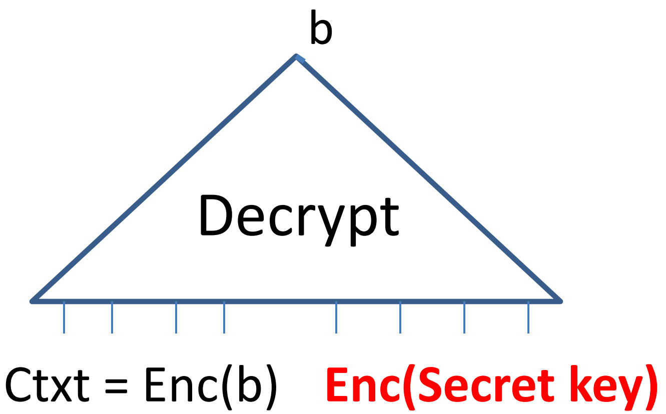
… I cannot release the secret key (lest everyone sees my data)

… but *I can release Enc(secret key)*



… called "Circular Encryption"

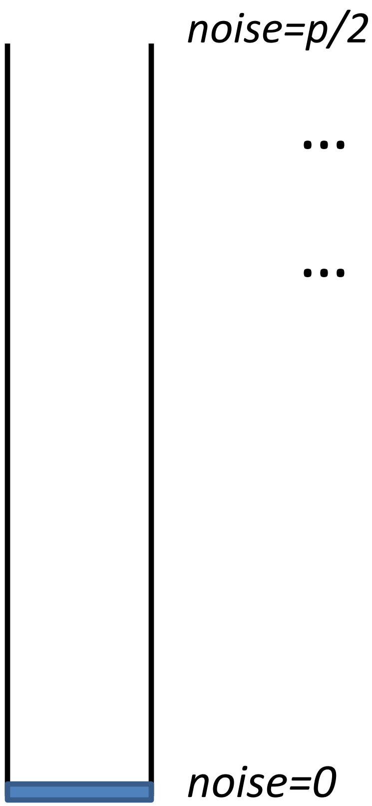




KEY IDEA:

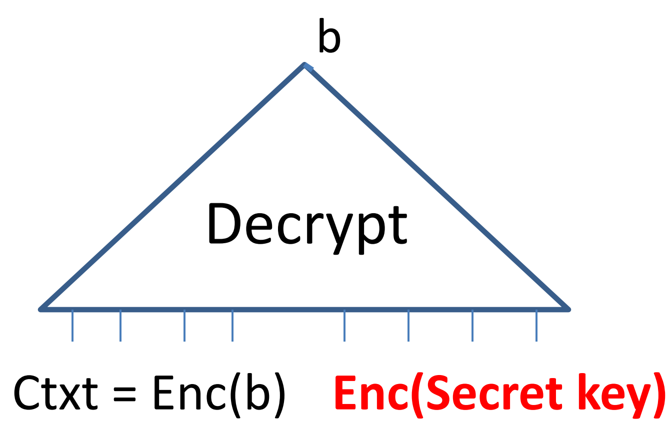
… I cannot release the secret key (lest everyone sees my data)

… but *I can release Enc(secret key)*



… Now, to reduce noise …

… Homomorphically evaluate the decryption ckt!!!

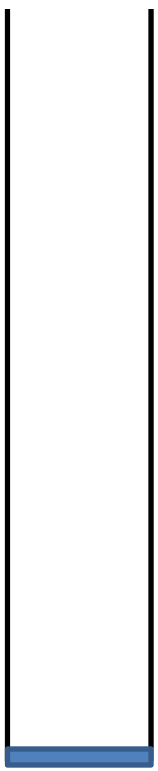


KEY IDEA:

... I cannot release the secret key (lest everyone sees my data)

... but *I can release Enc(secret key)*

noise=p/2

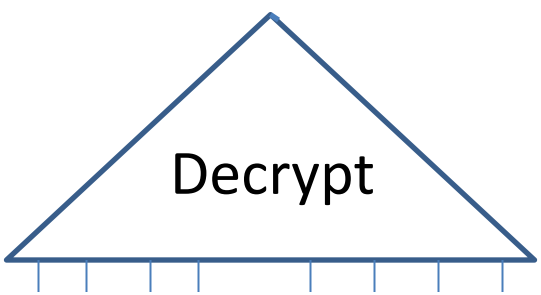


noise=0

... Now, to reduce noise ...

... Homomorphically evaluate the decryption ckt!!!

Enc(b)

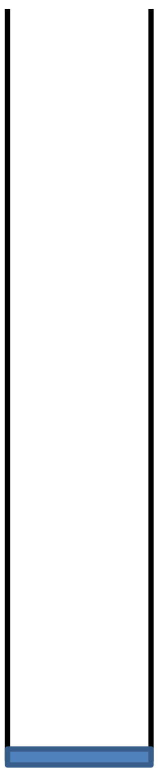


Ctxt = Enc(b) **Enc(Secret key)**

KEY IDEA:

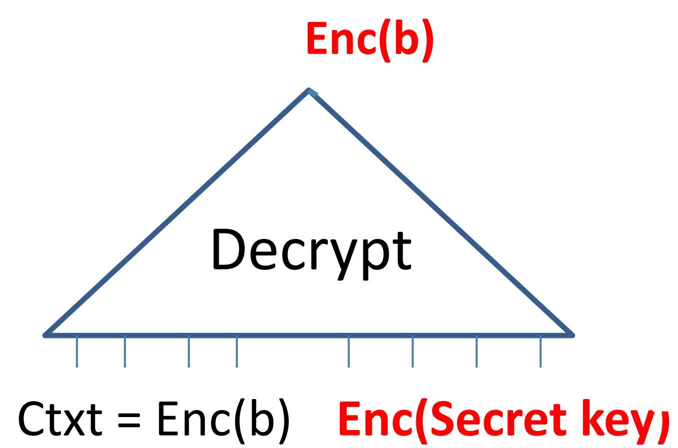
… I cannot release the secret key (lest everyone sees my data)

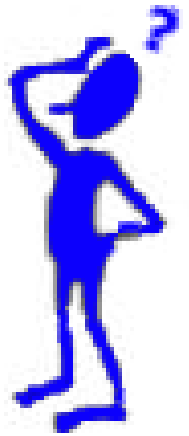
… but *I can release Enc(secret key)*



… Now, to reduce noise …

… Homomorphically evaluate the decryption ckt!!!

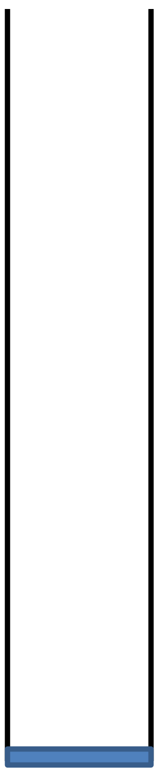




KEY IDEA:

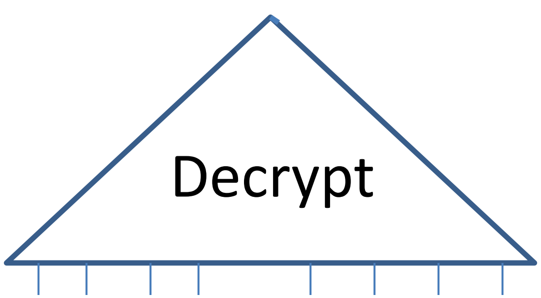
… I cannot release the secret key (lest everyone sees my data)

… but *I can release Enc(secret key)*



KEY OBSERVATION:

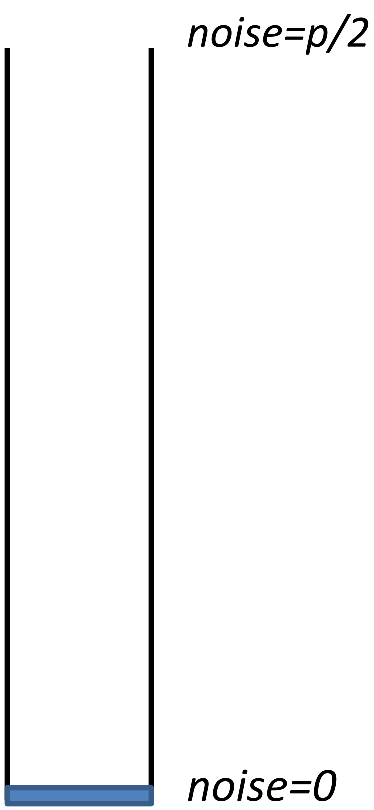
… the input Enc(b) and output Enc(b) have different noise levels …



KEY IDEA:

… I cannot release the secret key (lest everyone sees my data)

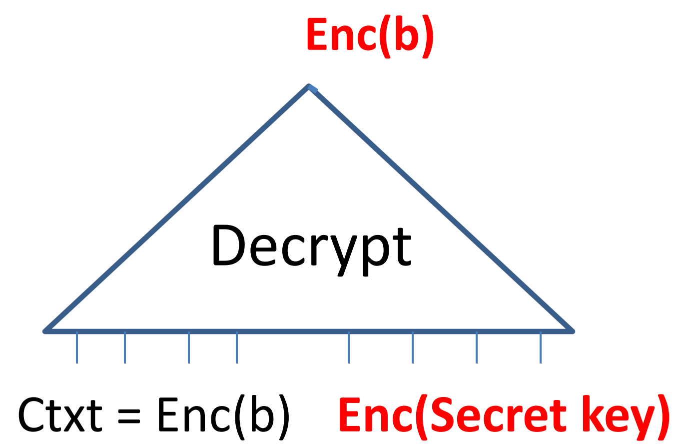
… but *I can release Enc(secret key)*



KEY OBSERVATION:

Regardless of the noise in the input Enc(b)…

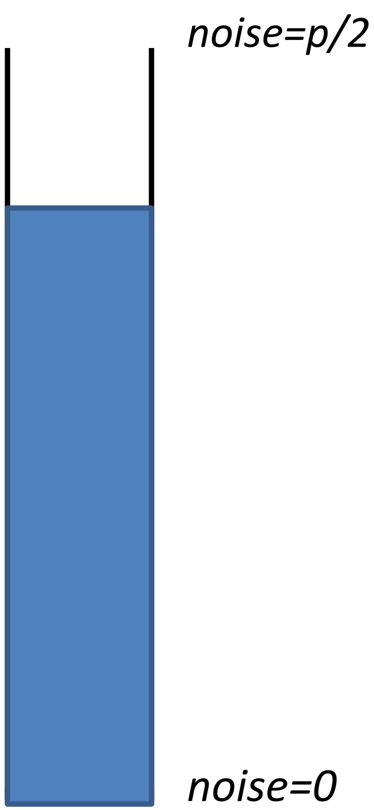
the noise level in the output Enc(b) is **FIXED**



KEY IDEA:

... I cannot release the secret key (lest everyone sees my data)

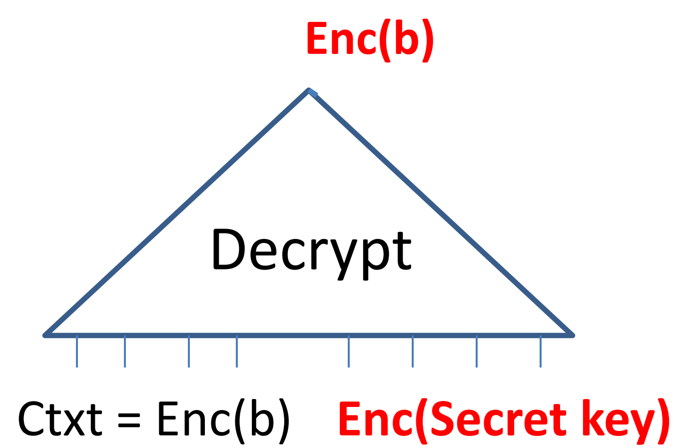
... but *I can release Enc(secret key)*



KEY OBSERVATION:

Regardless of the noise in the input Enc(b)...

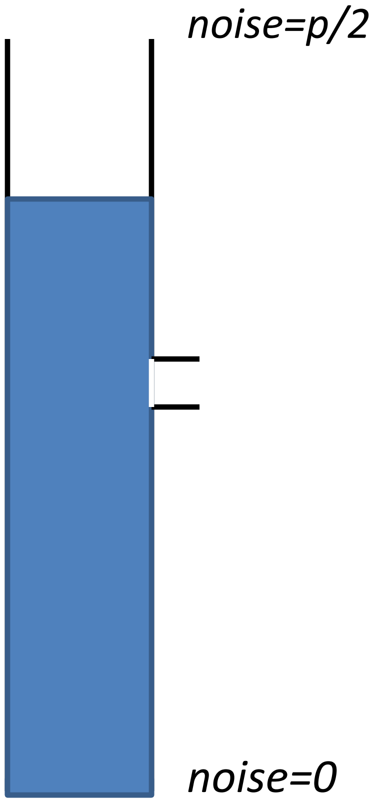
the noise level in the output Enc(b) is **FIXED**



KEY IDEA:

… I cannot release the secret key (lest everyone sees my data)

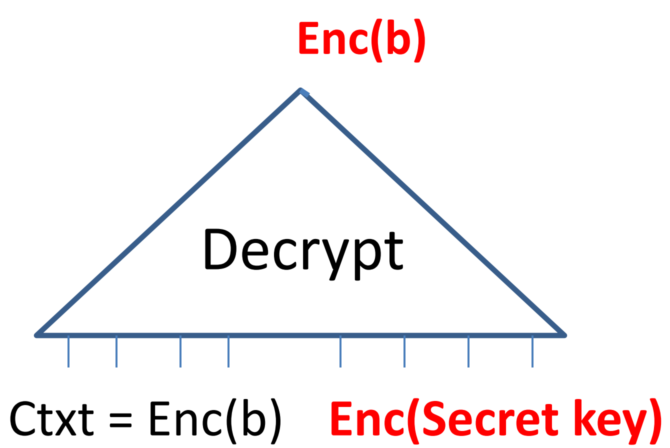
… but *I can release Enc(secret key)*



KEY OBSERVATION:

Regardless of the noise in the input Enc(b)…

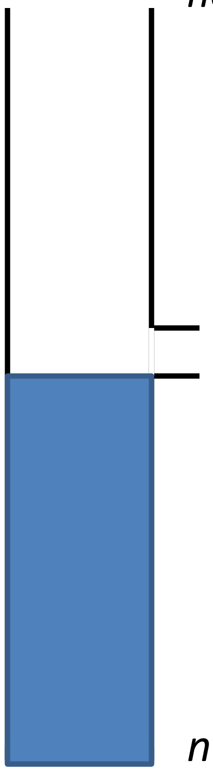
the noise level in the output Enc(b) is **FIXED**



KEY IDEA:

... I cannot release the secret key (lest everyone sees my data)

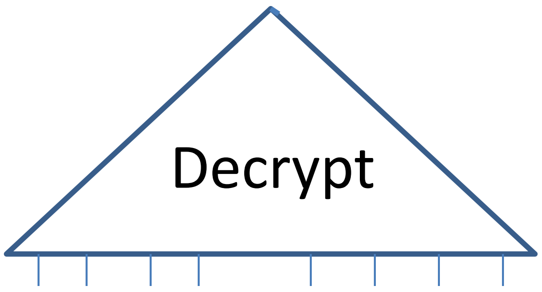
... but *I can release Enc(secret key)*



KEY OBSERVATION:

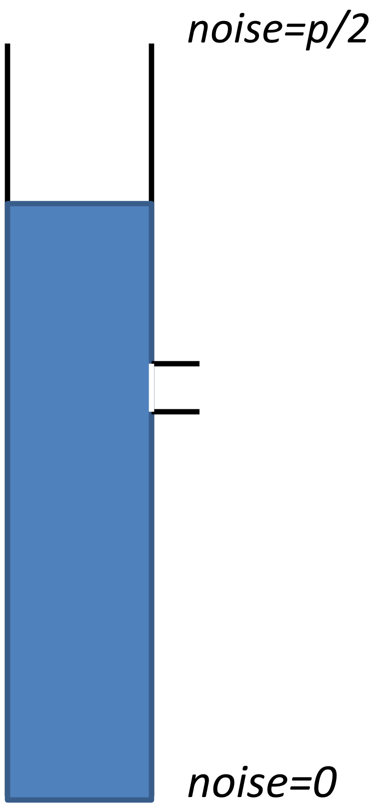
Regardless of the noise in the input Enc(b)...

the noise level in the output Enc(b) is **FIXED**

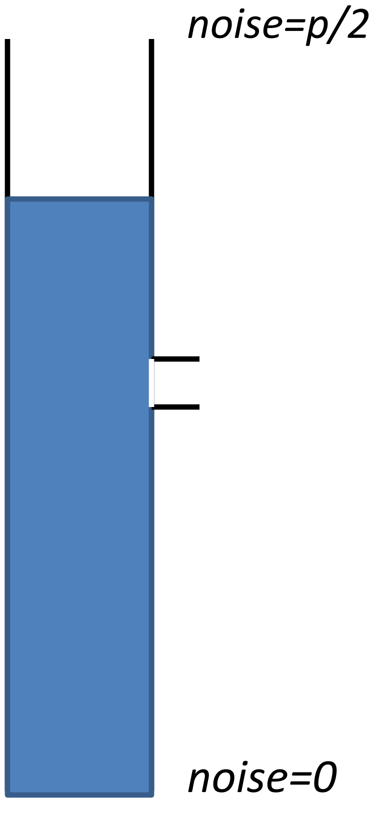


Bottomline: *whenever noise level increases beyond a limit …*

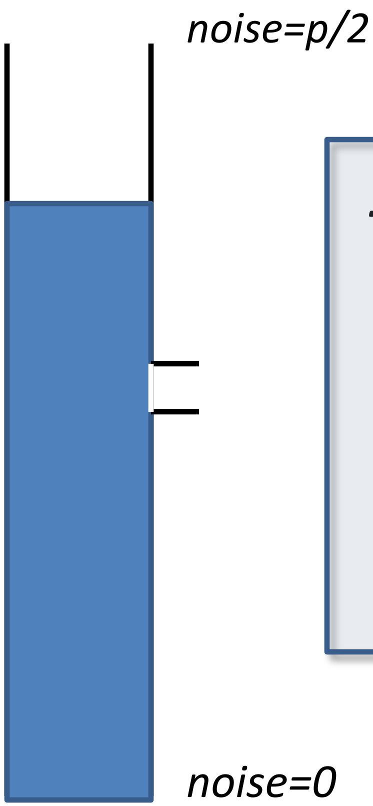
… use bootstrapping to reset it to a fixed level



Bootstrapping requires homomorphically evaluating the decryption circuit …



Bootstrapping requires homomorphically evaluating the decryption circuit …



*Thus, Gentry's **"bootstrapping theorem":***

If an enc scheme can evaluate its own decryption circuit, then it can evaluate everything