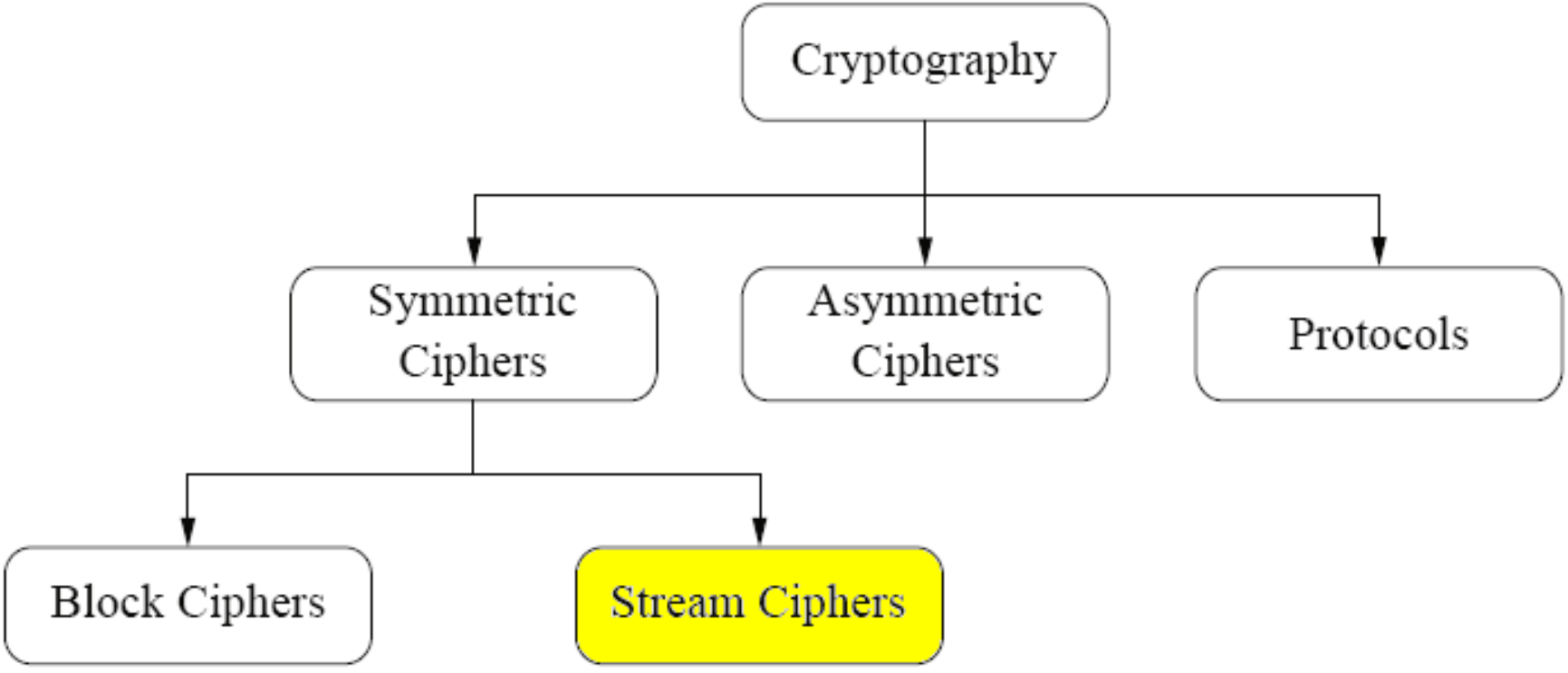
Introdução à Criptografia

Cifras de Fluxo e Criptoanálise

Prof. Rodrigo Minetto
rminetto@dainf.ct.utfpr.edu.br
Universidade Tecnológica Federal do Paraná
Material compilado de: Understanding Cryptography by Christof Paar e Jan Pelzl

Sumário

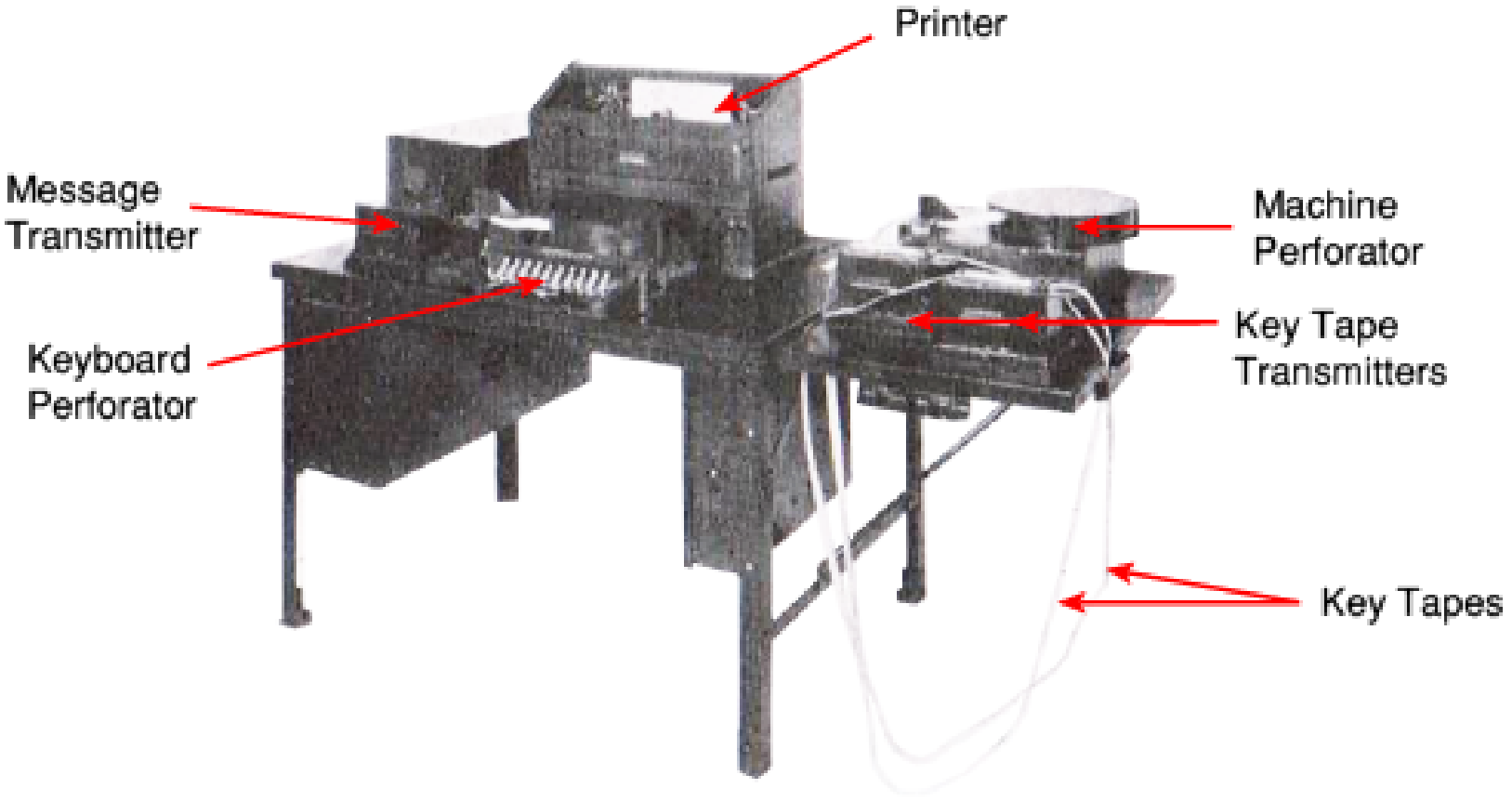
Algoritmos para criptografia



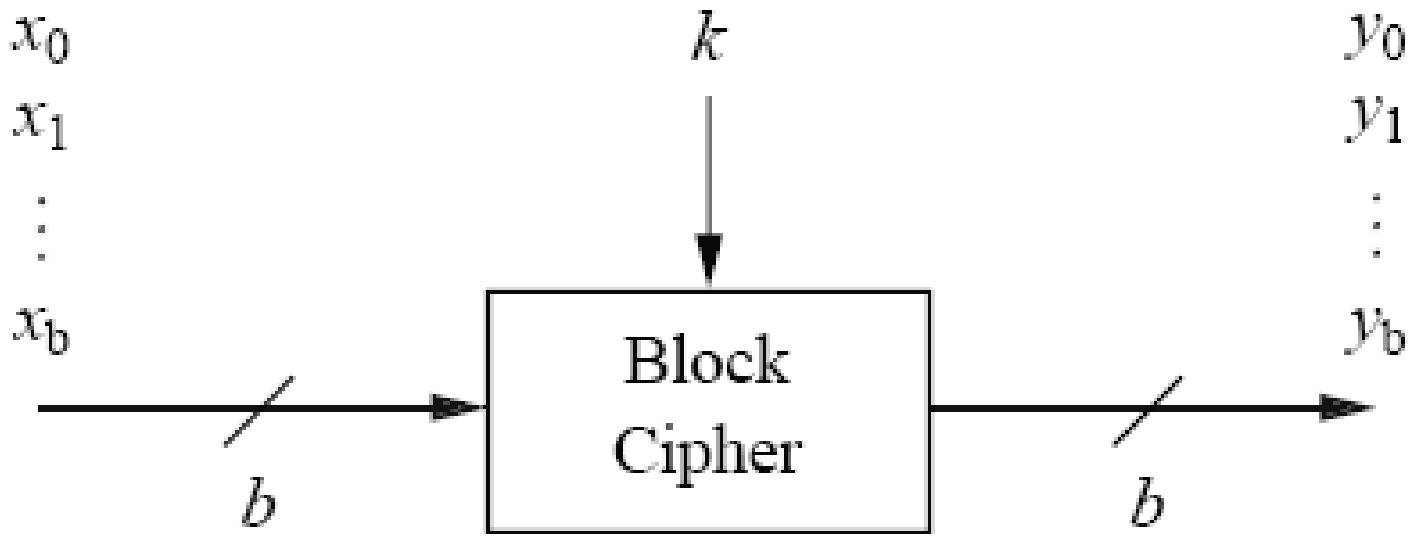
Cifras de fluxo

Cifras de fluxo foram patenteadas em 1917 por Gilbert Vernam. Ele construiu uma máquina eletro-mecânica que automaticamente cifrava comunicações de telégrafo. O texto em claro entrava como uma fita de papel e a chave como outra fita. Essa foi a primeira que vez que cifragem e comunição era feitas ao mesmo tempo por uma máquina.

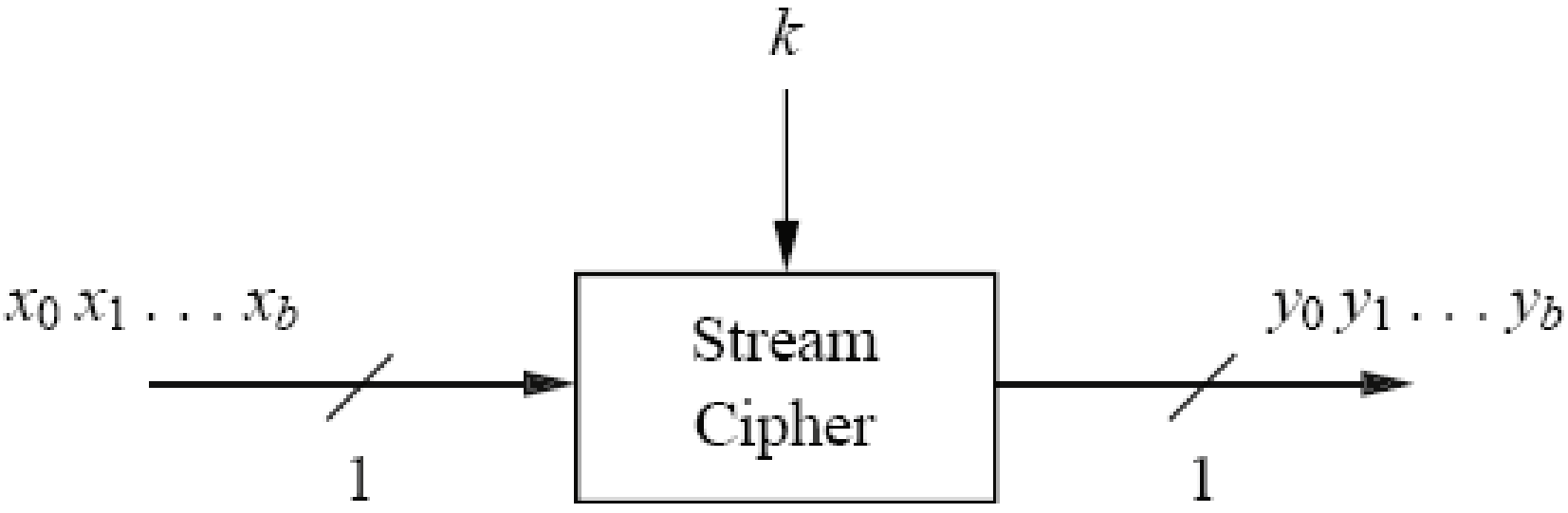
Máquina de Vernam



Cifra de fluxo vs Cifra de bloco



Cifra de fluxo vs Cifra de bloco



Cifras de fluxo

Observe que a **segurança** desse tipo de cifra depende totalmente do fluxo gerado para a chave. Assim, o algoritmo utilizado para gerar esse fluxo é basicamente a base de estudos para esse tipo de cifra. Os bits do fluxo de saída devem parecer uma sequência de números aleatórios para evitar ataques.

Cifras de fluxo

Cifra de fluxo

Sejam $x_i, y_i, k_i \in \mathbb{Z}_2$

Ciframento : $\mathbf{e}_{k_i}(x_i) = y_i \equiv x_i + k_i \ \mathbf{mod}\ 2.$

Deciframento: $\mathbf{d}_{k_i}(y_i) = x_i \equiv y_i + k_i \ \mathbf{mod}\ 2.$

tal que a chave secreta k_i é composta por apenas um bit, e **e** e **d** são as funções de cifragem e decifragem.

Cifras de fluxo

Porque o ciframento e deciframento usam a mesma função?

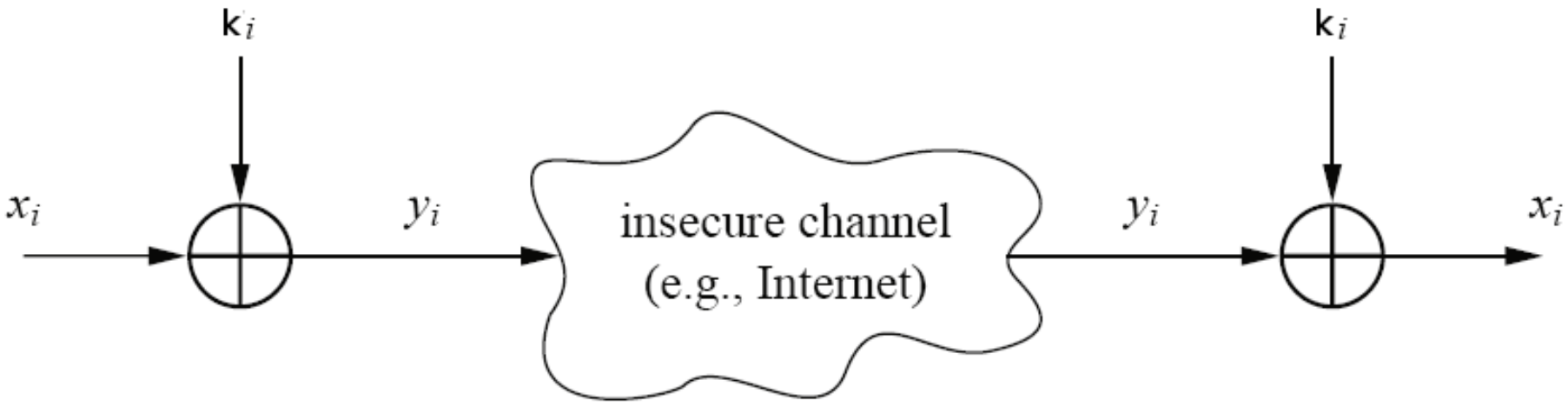
$$
\begin{aligned}
e_{k_i}(y) = y_i &\equiv x_i + k_i \bmod 2 \\
d_{k_i}(y) = x_i &\equiv y_i + k_i \bmod 2 \\
&\equiv (x_i + k_i) + k_i \bmod 2 \\
&\equiv x_i + k_i + k_i \bmod 2 \\
&\equiv x_i + 2k_i \bmod 2 \\
&\equiv x_i + 0 \bmod 2 \\
&\equiv x_i \bmod 2 \\
&\equiv x_i
\end{aligned}
$$

Cifras de fluxo

A aritmética módulo **2** produz como possíveis resultados os valores **0** ou **1**. Essa aritmética é equivalente a operação **XOR** $\oplus$.

x_i	k_i	$y_i \equiv x_i + k_i \bmod 2$
0	0	0
0	1	1
1	0	1
1	1	0

Cifra de fluxo com XOR



Cifras de fluxo

Porque XOR é uma função boa para cifra-mento?

x_i	k_i	y_i
0	0	0
0	1	1
1	0	1
1	1	0

Note que se a chave k_i for perfeitamente aleatória, então existe 50% de chance do re-sultado ser 0 ou 1 (balanceamento perfeito).

Cifras de fluxo

Porque **XOR** e não outra operação lógica como AND, OR ou NAND?

Alice	**Oscar**	**Bob**
$x_0, \ldots, x_6 = 1000001 = \mathtt{A}$		
$\oplus$		
$\mathsf{k}_0, \ldots, \mathsf{k}_6 = 0101100$		
$y_0, \ldots, y_6 = 1101101 = \mathtt{m}$		
	$\xrightarrow{\mathtt{m}}$	$y_0, \ldots, y_6 = 1101101$
		$\oplus$
		$\mathsf{k}_0, \ldots, \mathsf{k}_6 = 0101100$
		$x_0, \ldots, x_6 = 1000001 = \mathtt{A}$

Cifras de fluxo

Porque **XOR** e não outra operação lógica como AND, OR ou NAND? Resposta: **XOR é função inversível!**

Alice **Oscar** **Bob**

$x_0, \ldots, x_6 = 1000001 = \mathtt{A}$

$\oplus$

$\mathsf{k}_0, \ldots, \mathsf{k}_6 = 0101100$

$y_0, \ldots, y_6 = 1101101 = \mathtt{m}$

$\xrightarrow{\mathtt{m}}$

$y_0, \ldots, y_6 = 1101101$

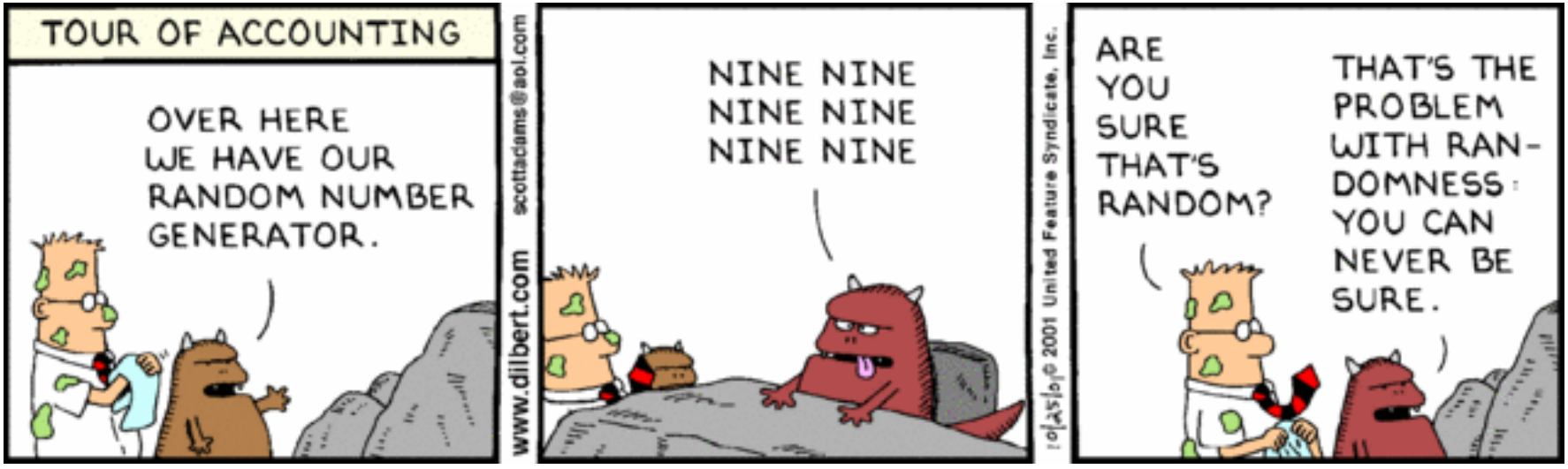
$\oplus$

$\mathsf{k}_0, \ldots, \mathsf{k}_6 = 0101100$

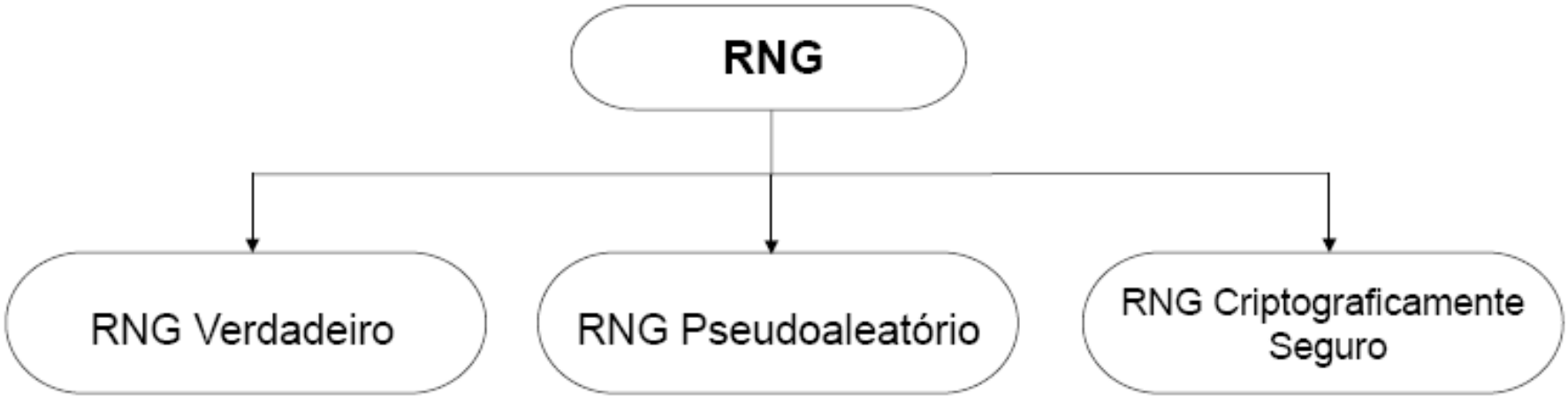
$x_0, \ldots, x_6 = 1000001 = \mathtt{A}$

Sumário

Números aleatórios



Números aleatórios

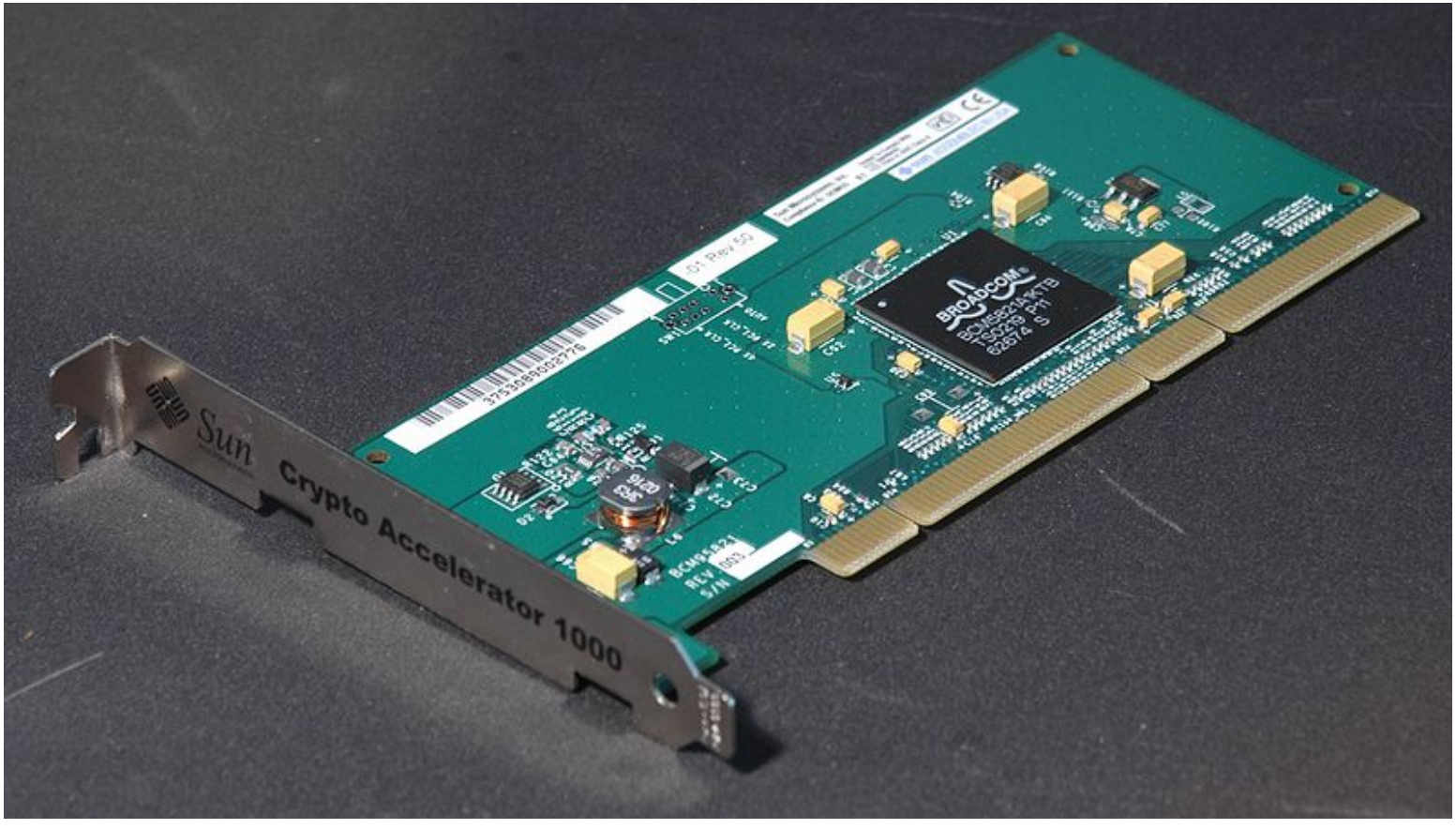


Números aleatórios

TRNG - geradores de números verdadeiramente aleatórios: são caracterizados pelo fato que sua saída não pode ser reproduzida. Por exemplo, lançar uma moeda 100 vezes e anotar o resultado como uma sequência de 100 bits. Nesse cenário, é praticamente impossível mais alguém na terra gerar a mesma sequência de 100 jogadas (ou bits). A chance de sucesso é $1/2^{100}$. TRNG são geralmente baseados em processos físicos: lançamento de dados, ruído de semicondutores, decaimento radiativo.

Cifras de fluxo

TRNG (produz números aleatórios por fenômenos físicos)



Números aleatórios

PRNG - geradores de números pseudo aleatórios: são caracterizados pela produção de uma sequência que é **calculada** baseada em uma *semente inicial*. São em geral recursivos e possuem a seguinte configuração

$$
\begin{aligned}
s_0 &= \text{seed} \\
s_{i+1} &= f(s_i) \qquad i = \{0, 1, \dots\}
\end{aligned}
$$

$$
\begin{aligned}
s_0 &= \text{seed} \\
s_{i+1} &\equiv as_i + b \bmod m, \qquad i = \{0, 1, \dots\}
\end{aligned}
$$

Números aleatórios

Observe que PRNG não são números verdadeiramente aleatórios, pois existe uma fórmula para determiná-los, o que os torna completamente determinísticos. Uma função PRNG amplamente utilizada é a *rand*() do ANSI C

$$
\begin{aligned}
s_0 &= 12345 \\
s_{i+1} &\equiv 1103515245\, s_i + 12345 \bmod 2^{31} \\
&\quad \text{para } i = \{0, 1, \ldots\}
\end{aligned}
$$

Números aleatórios

PRNG possuem boas propriedades estatísticas, isto é, a sequência de números produzida se aproxima de números verdadeiramente aleatórios. Existem testes matemáticos para verificar esse comportamento estatístico. PNRG são extremamente importantes para muitas aplicações (simulações e testes). Razão pela qual são incluídos em praticamente qualquer linguagem de programação.

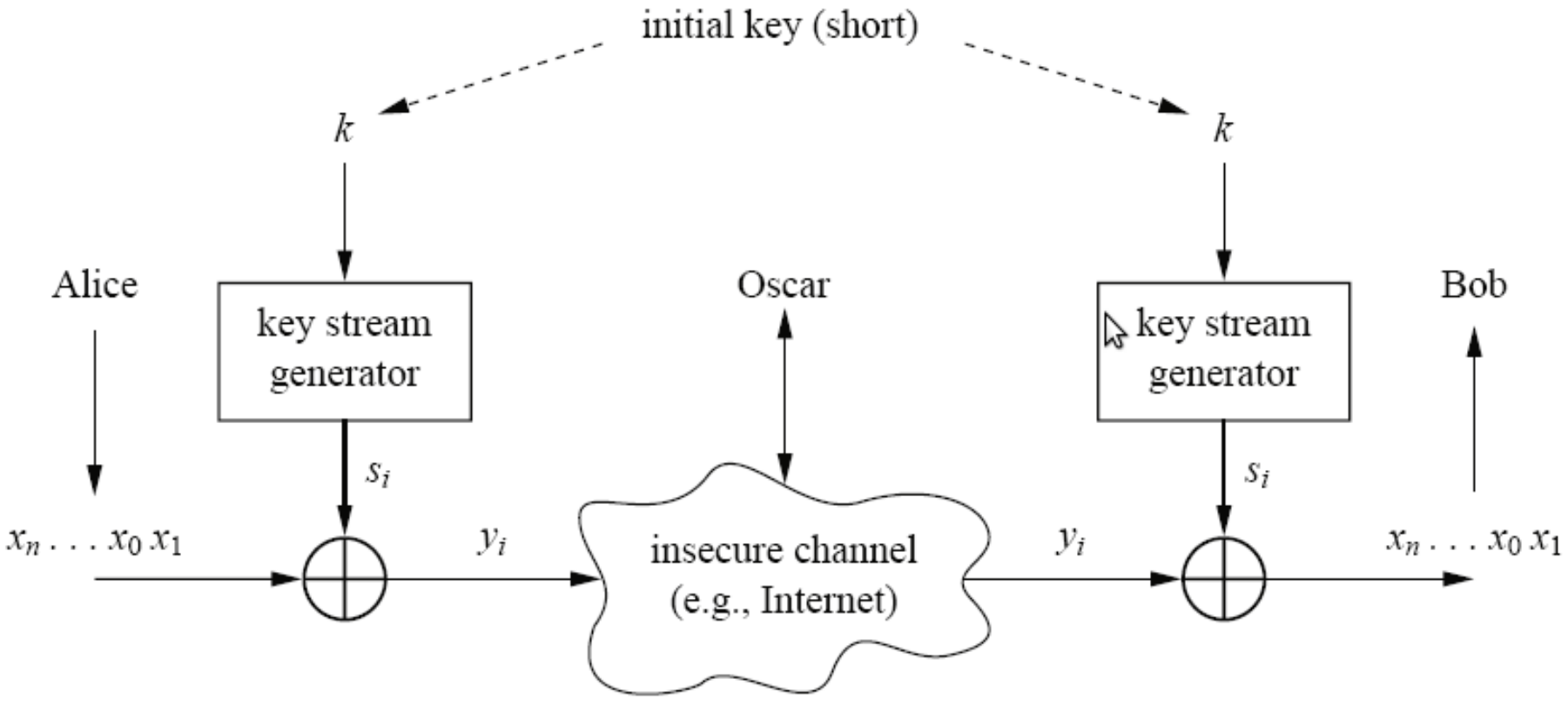
Números aleatórios

CSPRNG - geradores de números pseudo aleatórios seguros para criptografica: um CSPRNG é um PRNG imprevisível. Informalmente, isto significa que dado n bits de saída para um fluxo $s_i, s_{i+1}, \ldots, s_{i+n-1}$, tal que n é algum inteiro, é computacionalmente inviável determinar $s_{i+n}, s_{i+n+1}, \ldots$. Formalmente, dados n bits consecutivos para um fluxo, não existe algoritmo com tempo polinomial que pode prever o próximo bit s_{n+1} com chance maior que 50% de sucesso.

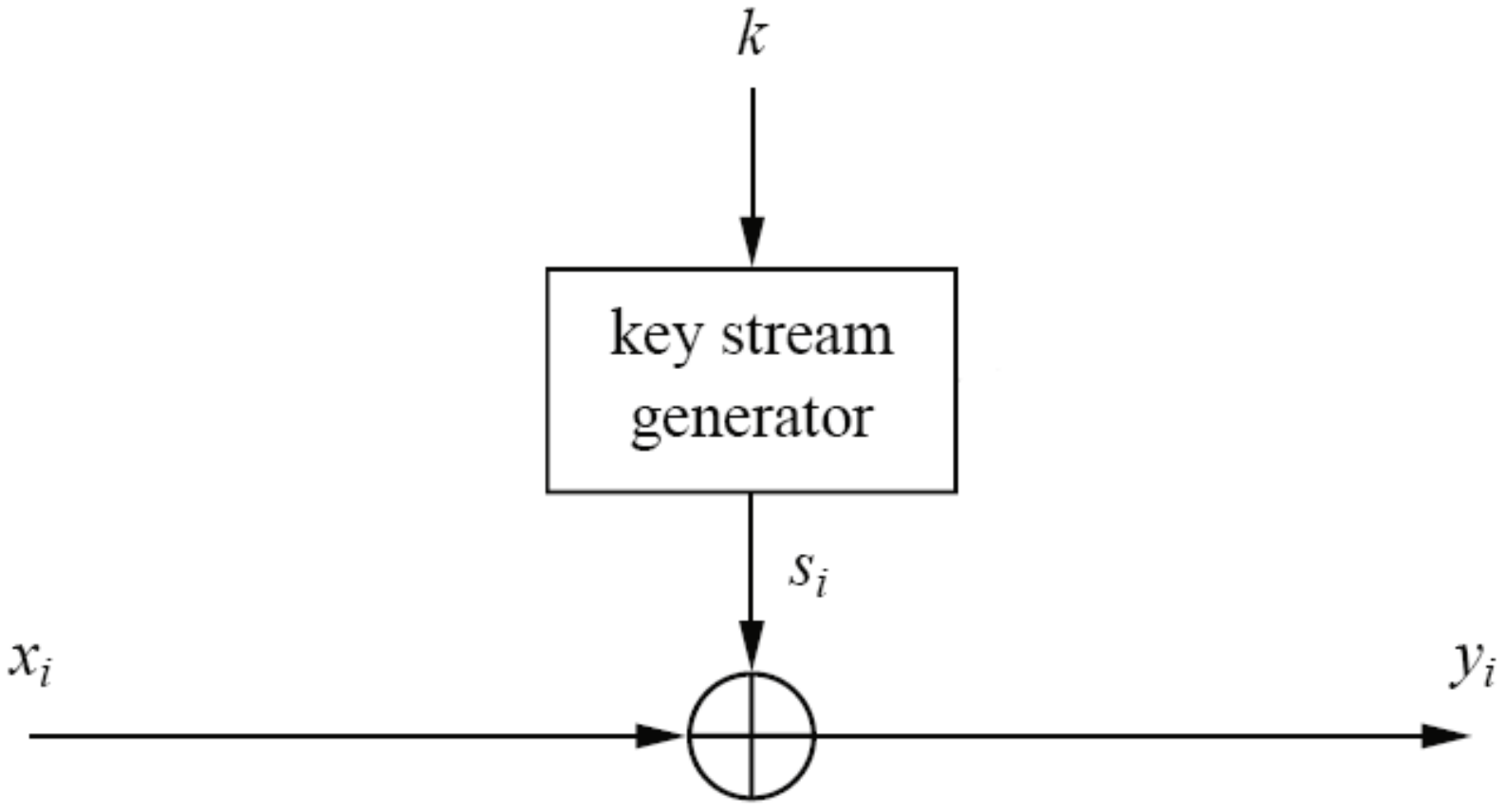
Números aleatórios

CSPRNG são necessários para a criptografia, sobretudo para cifras de fluxo. No entanto, exceto na criptografia, quase não há outras aplicações que necessitem de imprevisibilidade, enquanto muito e muitos sistemas necessitam de PRNG.

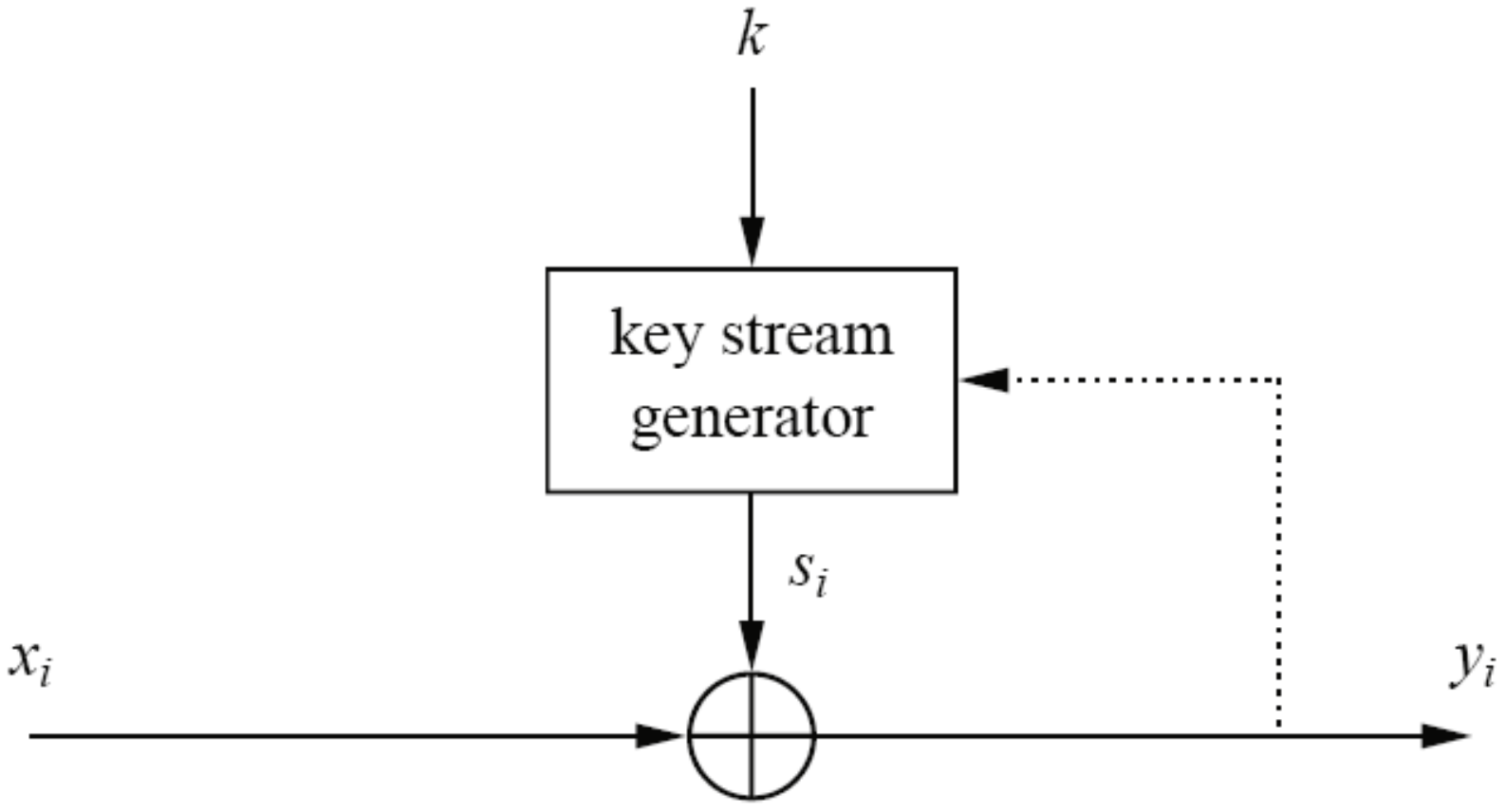
Cifras de fluxo



Cifra de fluxo síncrona



Cifra de fluxo assíncrona



Sumário

Cifra perfeita

Definição

Um **sistema criptográfico é incondicionalmente seguro** se ele não puder ser quebrado mesmo com recursos computacionais infinitos.

Cifra perfeita

Suponha um sistema simétrico com uma chave de 10.000 bits e que o único ataque conhecido é o por força bruta. Esse sistema é incondicionalmente seguro? Não, pois em teoria eu estou supondo recursos infinitos, assim, o atacante utiliza $2^{10.000}$ (número quase quatro vezes maior que o número de átomos do universo) computadores para verficar cada chave e a resposta é obtida em um passo. Uma cifra com essa característica é meramente conhecida como **computacionalmente segura**.

One-Time Pad

One-Time Pad (Cifra de uso único)

OTP é uma cifra de fluxo tal que:

- O fluxo da chave $k_0, k_1, k_2, \ldots$ é gerado por um **TRNG**.
- O fluxo $k_0, k_1, k_2, \ldots$ é conhecido somente pelas partes legítimas que se comunicam.
- Todo bit k_i somente é **utilizado uma vez**.

A cifra OTP é **incondicionalmente segura**.

Cifra perfeita

Porque a OTP é incondicionalmente seguro?

Para cada bit cifrado, produzimos uma das seguintes equações:

$$\begin{aligned} y_0 &= x_0 \oplus k_0 \bmod 2 \\ y_1 &= x_1 \oplus k_1 \bmod 2 \\ &\dots \end{aligned}$$

Cada equação linear tem duas incógnitas e para cada y_i, temos que $x_i = 0$ e $x_i = 1$ são equiprováveis! Se $k_0, k_1, \dots$ forem independentes e gerados por um TRNG, pode-se provar que esse sistema não pode ser resolvido.

Cifra perfeita

Desvantagem: para a maioria das aplicações o OTP é impraticável!

Sumário

Criptoanálise

Quais são dois problemas práticos com a cifra OTP?

- (1) produção de uma quantidade enorme de bits verdadeiramente aleatórios.

- (2) distribuição e proteção de chaves (para cada mensagem a ser enviada, é necessária uma chave de igual comprimento tanto para o emissor quanto para o receptor).

Criptoanálise

Observe que o número 0 é a **operação identidade** do XOR. Ou seja, o XOR de qualquer bit com o número 0, não altera o seu valor.

$x_i \oplus k_i$		y_i
0	0	0
0	1	1
1	0	1
1	1	0

Exemplo:

$$
\begin{array}{ll}
011111110001010 & \oplus \\
000000000000000 & \\
\hline
011111110001010 &
\end{array}
$$

Criptoanálise

Observe também que o XOR de um bit com ele próprio produz 0.

$x_i \oplus k_i$	y_i
0 0	0
0 1	1
1 0	1
1 1	0

Exemplo:

$$
\begin{array}{ll}
011111110001010 & \oplus \\
011111110001010 & \\
\hline
000000000000000 &
\end{array}
$$

Criptoanálise

Essas duas propriedades do XOR permitem:

$$
\begin{aligned}
&= \text{Texto cifrado} \oplus \text{Chave} \\
&= (\text{Texto original} \oplus \text{Chave}) \oplus \text{Chave} \\
&= \text{Texto original} \oplus (\text{Chave} \oplus \text{Chave}) \\
&= \text{Texto original} \oplus 0\text{x}00\ldots00 \\
&= \text{Texto original}
\end{aligned}
$$

Criptoanálise

Many-time pad: é uma cifra que funciona exatamente como a one-time pad, com a exceção de que as **chaves não são únicas** para cada mensagem. Esse reaproveitamento de chaves elimina a característica de incondicionalmente segura da OTP.

Criptoanálise

Um ataque conhecido é utilizado quando duas ou mais mensagens são cifradas com a mesma chave. O XOR das duas mensagens **remove a chave** e o resultado é o XOR entre os dois textos originais:

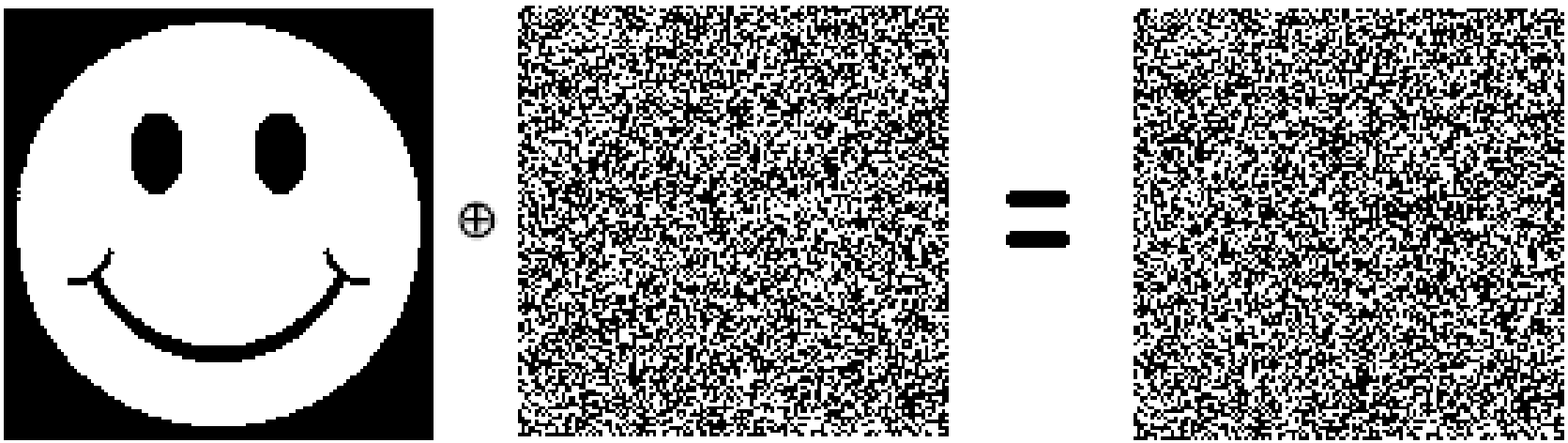
$$
\begin{aligned}
&= (\text{Texto A} \oplus \text{Chave}) \oplus (\text{Texto B} \oplus \text{Chave}) \\
&= (\text{Texto A} \oplus \text{Texto B}) \oplus (\text{Chave} \oplus \text{Chave}) \\
&= (\text{Texto A} \oplus \text{Texto B}) \oplus 0\text{x}00\ldots00 \\
&= (\text{Texto A} \oplus \text{Texto B})
\end{aligned}
$$

Criptoanálise

$$M_1 \oplus K = C_1$$

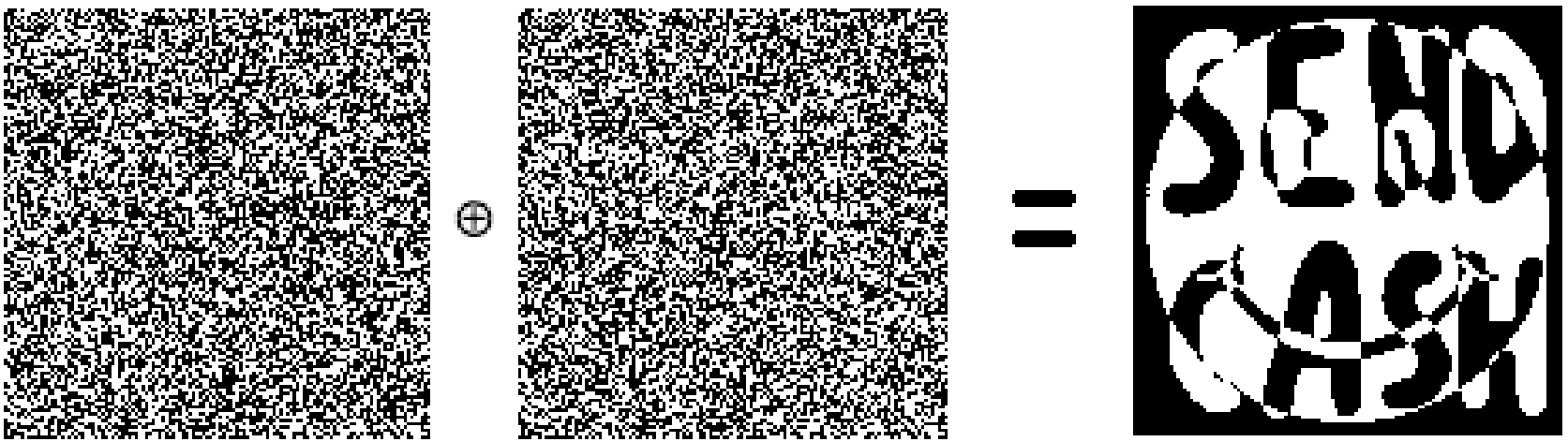
Criptoanálise

$$M_2 \oplus K = C_2$$



Criptoanálise

$$C_1 \oplus C_2 = M_1 \oplus K \oplus M_2 \oplus K = M_1 \oplus M_2$$



Criptoanálise

Mesmo que nenhuma das mensagens seja conhecida, desde que se saiba que as duas mensagens estão em linguagem natural ou que possuem cabeçalhos de pacotes ip's, termos de negócios ou transações, então um ataque pode ser realizado com sucesso. Se uma das mensagens for conhecida, a solução é trivial.

Criptoanálise

Para a criptoanálise da cifra **many-time pad** vamos supor um gerador de números aleatórios com uma mesma semente inicial.

```
int aleatorio (void) {
  next = next * 1103515245 + 12345;
  return (next % 0x7fffffff);
}

void semente (unsigned seed) {
  next = seed;
}
```

Criptoanálise

Suponha o envio de duas mensagens:

M_1: um dia que termina hoje

M_2: outro que comeca amanha

cifradas com a **MESMA** chave (ou seja, com a mesma semente inicial).

Criptoanálise

M_1	u	m	d	i	a	q	u	e	t	e	r	m	i	n	a	h	o	j	e
D	20	12	03	08	00	16	20	04	19	04	17	12	08	13	00	07	14	09	04
H	14	0c	03	08	00	10	14	04	13	04	11	0c	08	0d	00	07	0e	09	04

D: código decimal extraído de M_1 (a = 00, ..., z = 25).

H: código em hexadecimal extraído de M_1.

Criptoanálise

M_1	u	m	d	i	a	q	u	e	t	e	r	m	i	n	a	h	o	j	e
D	20	12	03	08	00	16	20	04	19	04	17	12	08	13	00	07	14	09	04
K	20	12	14	11	09	06	06	01	12	02	23	21	25	18	09	22	10	10	09

D: código decimal extraído de M_1 (a = 00, ..., z = 25).

K: chave aleatória (semente = 512).

Criptoanálise

M_1	u	m	d	i	a	q	u	e	t	e	r	m	i	n	a	h	o	j	e
D	20	12	03	08	00	16	20	04	19	04	17	12	08	13	00	07	14	09	04
K	20	12	14	11	09	06	06	01	12	02	23	21	25	18	09	22	10	10	09
$\oplus$	00	00	13	03	09	22	18	05	31	06	06	25	17	31	09	17	04	03	13
C_1	a	a	n	d	j	w	s	f	D	g	g	z	r	D	j	r	e	d	n

D: código decimal de M_1 (a $= 0$, ..., z $= 25$).

K: chave aleatória (semente $= 512$).

$\oplus$: $D \oplus K$ (XOR)

C_1: texto cifrado de M_1.

Criptoanálise

M_2	o	u	t	r	o	q	u	e	c	o	m	e	c	a	a	m	a	n	h	a
D	14	20	19	17	14	16	20	04	02	14	12	04	02	00	00	12	00	13	07	00
C	20	12	14	11	09	06	06	01	12	02	23	21	25	18	09	22	10	10	09	16
$\oplus$	26	24	29	26	07	22	18	05	14	12	27	17	27	18	09	26	10	07	14	16
C_2	{	y	~	{	h	w	s	f	o	m	|	r	|	s	j	{	k	h	o	q

D: código decimal de M_2 (a $= 0$, ..., z $= 25$).

K: chave aleatória (semente $= 512$).

$\oplus$: $D \oplus K$ (XOR)

C_2: texto cifrado de M_2

Criptoanálise

Alinhando strings:

	u	m	d	i	a	q	u	e	t	e	r	m	i	n	a	h	o	j	e
D	20	12	03	08	00	16	20	04	19	04	17	12	08	13	00	07	14	09	04
K	20	12	14	11	09	06	06	01	12	02	23	21	25	18	09	22	10	10	09
$\oplus$	00	00	13	03	09	22	18	05	31	06	06	25	17	31	09	17	04	03	13
	o	u	t	r	o	q	u	e	c	o	m	e	c	a	a	m	a	n	h
D	14	20	19	17	14	16	20	04	02	14	12	04	02	00	00	12	00	13	07
K	20	12	14	11	09	06	06	01	12	02	23	21	25	18	09	22	10	10	09
$\oplus$	26	24	29	26	07	22	18	05	14	12	27	17	27	18	09	26	10	07	14

D: código decimal.

K: chave aleatória (semente $= 512$).

$\oplus$: $D \oplus K$ (XOR)

Criptoanálise

Textos cifrados alinhados:

M_1	u	m	d	i	a	q	u	e	t	e	r	m	i	n	a	h	o	j	e
C_1	a	a	n	d	j	w	s	f	D	g	g	z	r	D	j	r	e	d	n
$\oplus$	00	00	13	03	09	22	18	05	31	06	06	25	17	31	09	17	04	03	13
M_2	o	u	t	r	o	q	u	e	c	o	m	e	c	a	a	m	a	n	h
C_2	{	y	~	{	h	w	s	f	o	m	|	r	|	s	j	{	k	h	o
$\oplus$	26	24	29	26	07	22	18	05	14	12	27	17	27	18	09	26	10	07	14

$\oplus$ = Texto original $\oplus$ Chave (semente = 512).

Criptoanálise

Ataque: XOR dos textos cifrados C_1 e C_2 :

C_1	a	a	n	d	j	w	s	f	D	g	g	z	r	D	j	r	e	d	n
	00	00	13	03	09	22	18	05	31	06	06	25	17	31	09	17	04	03	13
C_2	{	y	∼	{	h	w	s	f	o	m	|	r	|	s	j	{	k	h	o
	26	24	29	26	07	22	18	05	14	12	27	17	27	18	09	26	10	07	14
$C_1 \oplus C_2$	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
$C_1 \oplus C_2$	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03

$\oplus$ = (Texto A $\oplus$ Chave) $\oplus$ (Texto B $\oplus$ Chave)

Criptoanálise

Ataque: **Crib dragging**

- 1) Suponha uma palavra que pode aparecer em uma das mensagens (palpite).
- 2) Faça um XOR dos dois textos cifrados.
- 3) Faça um XOR do palpite em 1) com cada posição da string gerada pelo passo 2).
- 4) Se o resultado em 3) for legível, descubra a palavra e expanda o crig.
- 5) Se o resultado não for legível, faça um XOR do palpite com a próxima posição.

Criptoanálise

Suposição (palavra): “**que**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
	16	20	04																
	q	u	e																
⊕	10	12	20																
	k	m	u																

Criptoanálise

Suposição (palavra): “**que**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
		16	20	04															
		q	u	e															
⊕		08	04	29															
		i	e	∼															

Criptoanálise

Suposição (palavra): “**que**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
			16	20	04														
			q	u	e														
⊕			00	13	10														
			a	n	k														

Criptoanálise

Suposição (palavra): “**que**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
				16	20	04													
				q	u	e													
⊕				09	26	04													
				j	{	e													

Criptoanálise

Suposição (palavra): “**que**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
					16	20	04												
					q	u	e												
⊕					30	20	04												
					D	u	e												

Criptoanálise

Suposição (palavra): “**que**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
						16	20	04											
						q	u	e											
⊕						16	20	04											
						q	u	e											

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
	19	04	17	12	08	13	00												
	t	e	r	m	i	n	a												
⊕	09	28	01	21	06	13	00												
	j	}	b	v	g	n	a												

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
		19	04	17	12	08	13	00											
		t	e	r	m	i	n	a											
⊕		11	20	08	02	08	13	00											
		l	u	i	c	i	n	a											

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
			19	04	17	12	08	13	00										
			t	e	r	m	i	n	a										
⊕			03	29	31	12	08	13	17										
			d	∼	D	m	i	n	r										

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
				19	04	17	12	08	13	00									
				t	e	r	m	i	n	a									
⊕				10	10	17	12	08	28	10									
				k	k	r	m	i	}	k									

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
					19	04	17	12	08	13	00								
					t	e	r	m	i	n	a								
⊕					29	04	17	12	25	07	29								
					∼	e	r	m	z	h	∼								

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
						19	04	17	12	08	13	00							
						t	e	r	m	i	n	a							
⊕						19	04	17	29	02	16	08							
						t	e	r	∼	c	q	i							

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
							19	04	17	12	08	13	00						
							t	e	r	m	i	n	a						
⊕							19	04	00	06	21	05	10						
							t	e	a	g	v	f	k						

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
								19	04	17	12	08	13	00					
								t	e	r	m	i	n	a					
⊕								19	21	27	17	00	07	13					
								t	v	|	r	a	h	n					

Criptoanálise

Suposição (palavra): “**termina**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
									19	04	17	12	08	13	00				
									t	e	r	m	i	n	a				
⊕									02	14	12	04	02	00	00				
									c	o	m	e	c	a	a				

Criptoanálise

Suposição: “**outro**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
	14	20	19	17	14														
⊕																			

Criptoanálise

Suposição: “**outro**”

	{	y	q	z	o	a	a	a	r	k	∼	i	k	n	a	l	o	e	d
	26	24	16	25	14	00	00	00	17	10	29	08	10	13	00	11	14	04	03
	14	20	19	17	14														
	o	u	t	r	o														
⊕	20	12	03	08	00														
	u	m	d	i	a														

Criptoanálise

Como recuperar a chave utilizada?

Chave = Cifrado $\oplus$ Suposição

M_2	o	u	t	r	o	q	u	e	c	o	m	e	c	a	a	m	a	n	h
C_2	{	y	~	{	h	w	s	f	o	m	|	r	|	s	j	{	k	h	o
C_2	26	24	29	26	07	22	18	05	14	12	27	17	27	18	09	26	10	07	14
	o	u	t	r	o														
	14	20	19	17	14														
$\oplus$	**20**	**12**	**14**	**11**	**09**														

Sequência: 20, 12, 14, 11, 09, ...

Criptoanálise

Como saber qual a semente utilizada?

```
void semente (unsigned seed) {
  next = seed;
}
```

```
seed =   0, seq = {21, 15,  2, 23, 25, ...}
seed =   1, seq = {22,  6,  0,  0, 25, ...}
seed =   2, seq = { 0, 20,  0,  3,  0, ...}
...
seed = 512, seq = {20, 12, 14, 11,  9, ...}
```

Como saber qual a semente utilizada?

Força-bruta: teste um milhão de chaves (1, 2, 3, 4, 5, 6, ..., 1.000.000) na função `semente`. A chave que produzir exatamente a sequência $\{20, 12, 14, 11, 09, \ldots\}$ é a que foi utilizada como semente.

Criptoanálise

Cifras de fluxo são maleáveis: mesmo a cifra OTP (incondicionalmente segura) não oferece integridade. Por exemplo, suponha uma mensagem m cifrada por Alice com k_a, ou seja $c_a = m \oplus k_a$. Oscar pode modificar a mensagem sem conhecer o conteúdo $c_o = c_a \oplus k_o$. No deciframento ocorre que $m \oplus k_o = (((m \oplus k_a) \oplus k_o) \oplus k_a)$ (ataque man-in-the-middle).