Secure Password-Based Cipher Suite for TLS

Michael Steiner
Saarland University, Saarbrücken, Germany
steiner@acm.org

Peter Buhler, Thomas Eirich, Michael Waidner
IBM Research, Rüschlikon, Switzerland
{bup,eir,wmi}@zurich.ibm.com

ISOC Network and Distributed Systems Security Symposium,
San Diego, February, 2000.

On Password-Based Authentication . . .

"Old fashioned ... "

"Esoteric toy for researchers . . ."

"Obsoleted by public key crypto and PKI?"

"Who cares?"

"Dictionary attacks!!"

"SSL handles that already ..."

"Insecure?!"

"Protocols don't scale!"

"Trivial !!"

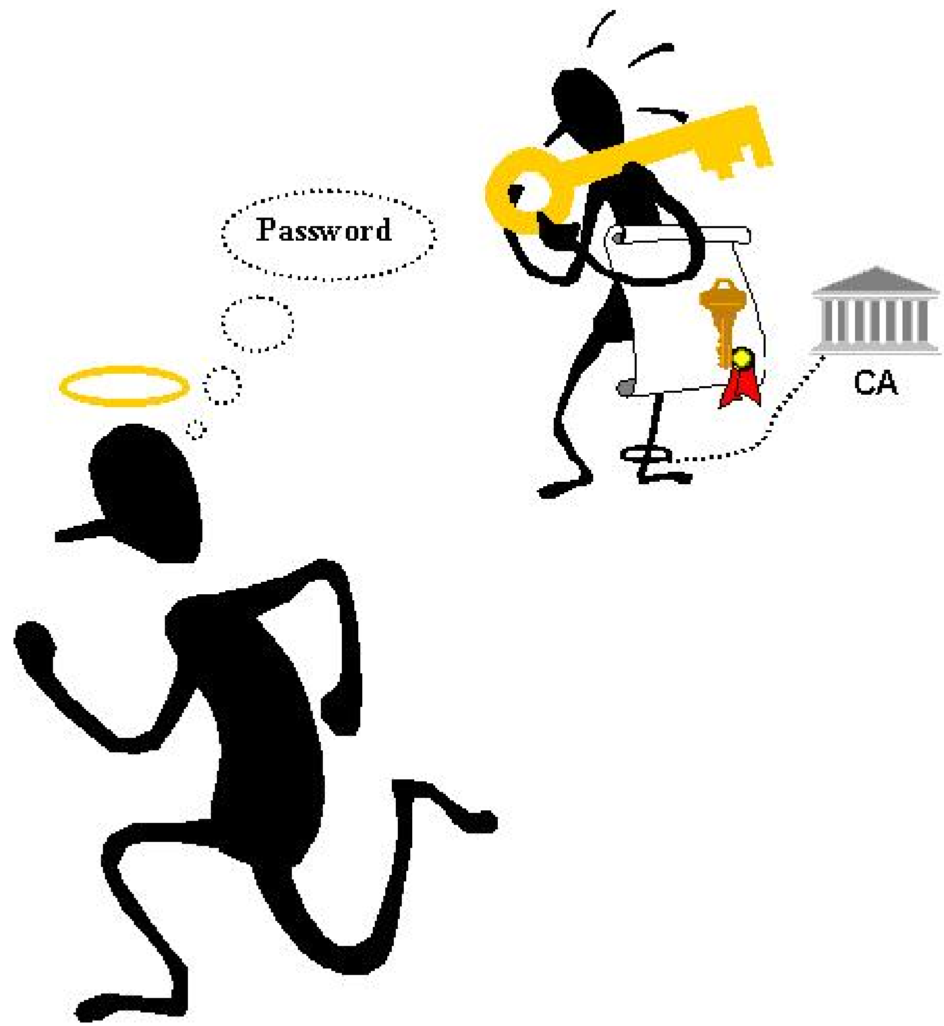
Password-based Protocols: Lightweight And Secure!

Advantages

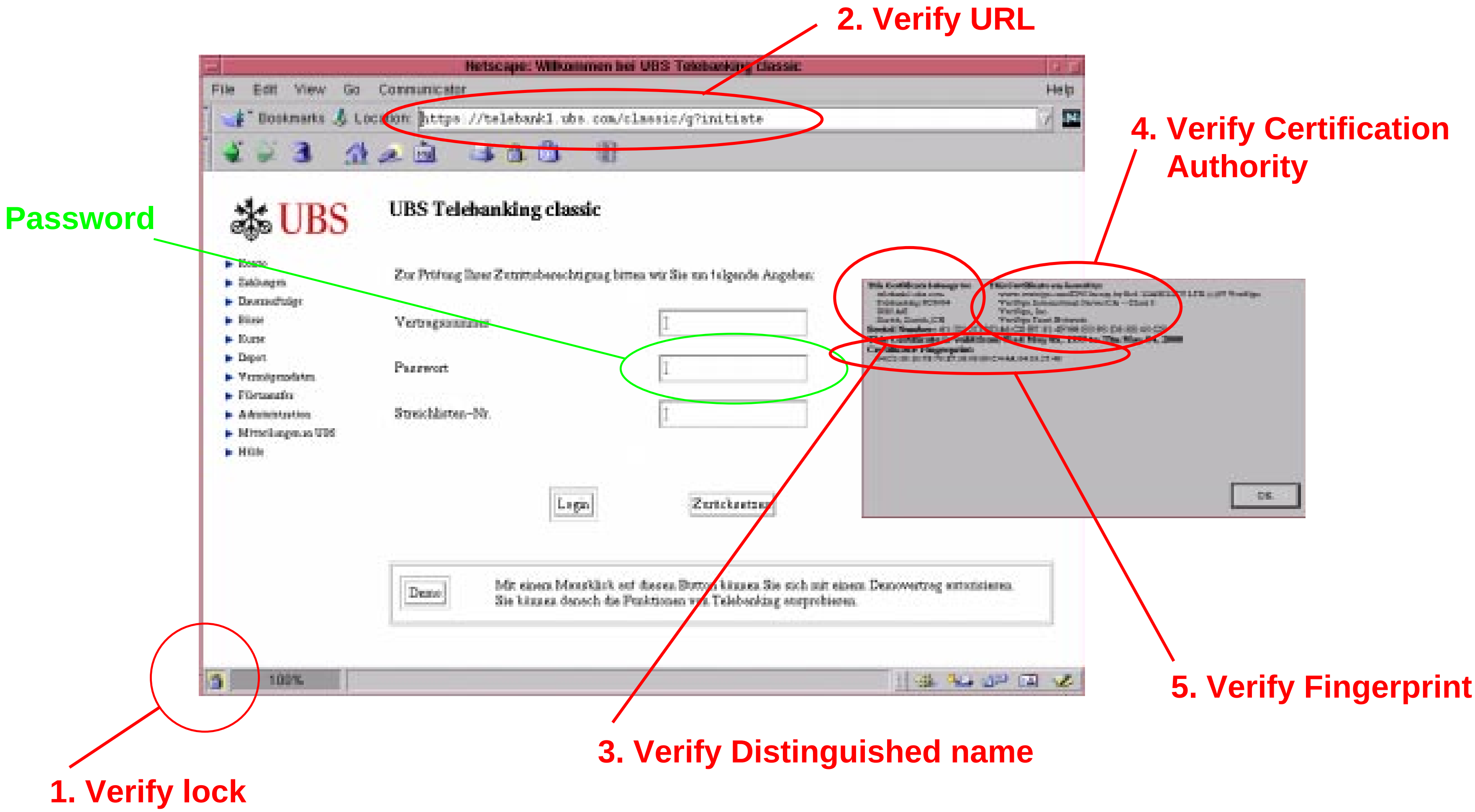
- No (heavy) infrastructure
- No (trusted) storage
- Maximal security
- Tolerates “clueless” users

Application

- Mobile Environments
- Bootstrapping
- Webbanking
- ...

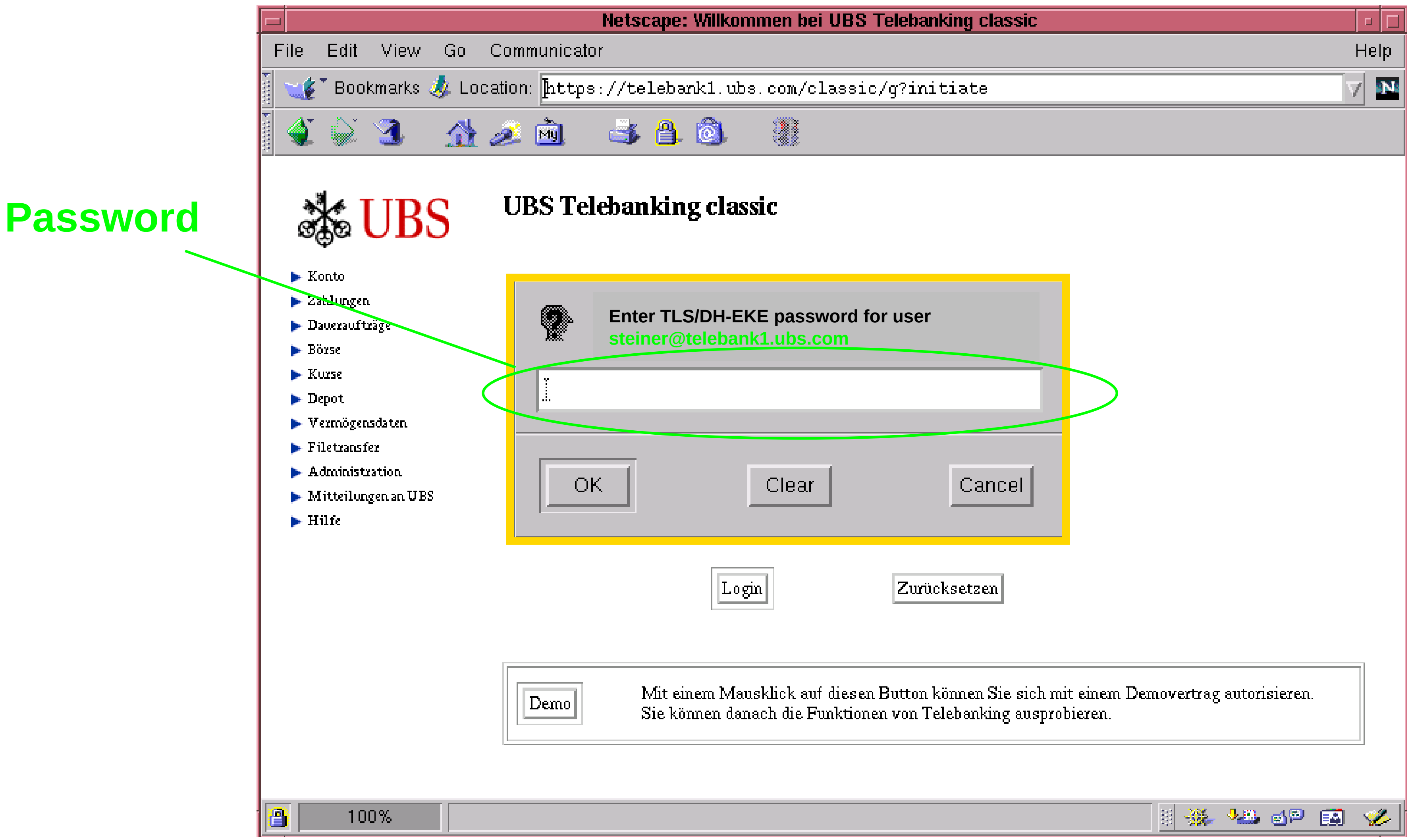


Sending Passwords On One-Way Authenticated SSL Channels



The user's (too) heavy burden . . .

SSL Channels With Passwords In Ideal World



Notation

p, q	Primes; $q \mid \phi(p)$
g	Generator in $\mathbb{Z}_p^*$
h	Generator in subgroup G of $\mathbb{Z}_p^*$ with order q
x, y	Secret exponent $\in_{\mathcal{R}} \mathbb{Z}_q$
$\boldsymbol{pwd}$	Password / weak secret
$E_{\boldsymbol{pwd}}$	Symmetric encryption with password as shared key
$\mathrm{MAC}(k, \dots)$	Message authentication code on ... with key k
$\mathcal{H}_x$	Pseudo-random functions
$\mathcal{G}_x$	Key derivation functions
k_{mstr}	master key for a session
k_{conf}	handshake confirmation key
k_{sess}	session key

Diffie-Hellman Encrypted Key Exchange (DH-EKE)

Client		Server
(password $\boldsymbol{pwd}$)		(password $\boldsymbol{pwd}$)
$x \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q$		
	$E_{\boldsymbol{pwd}}(h^x)$ $\longrightarrow$	
		$y \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q$
		$k_{mstr} \leftarrow (h^x)^y$
		$k_{conf} \leftarrow \mathcal{G}_1(k_{mstr})$
		$k_{sess} \leftarrow \mathcal{G}_2(k_{mstr})$
	$\longleftarrow$ $h^y, \mathrm{MAC}(k_{conf}, \text{“1”}, h^x, h^y)$	
$k_{mstr} \leftarrow (h^y)^x$		
$k_{conf} \leftarrow \mathcal{G}_1(k_{mstr})$		
$k_{sess} \leftarrow \mathcal{G}_2(k_{mstr})$		
abort if MAC not OK		
	$\mathrm{MAC}(k_{conf}, \text{“2”}, h^x, h^y)$ $\longrightarrow$	
		abort if MAC not OK

Problems With Encryption And Choice Of Algebraic Group

Encryption as verifier of password guesses

$$\text{accept } guess \text{ if } \mathrm{Test}(guess, enc) = \mathrm{OK}$$

$h^x \in$ multplicative group $\mathbb{Z}_p^*$

- Dense mapping $\Rightarrow$ encryption "ok" ...
- ... but random oracles required for proof of security

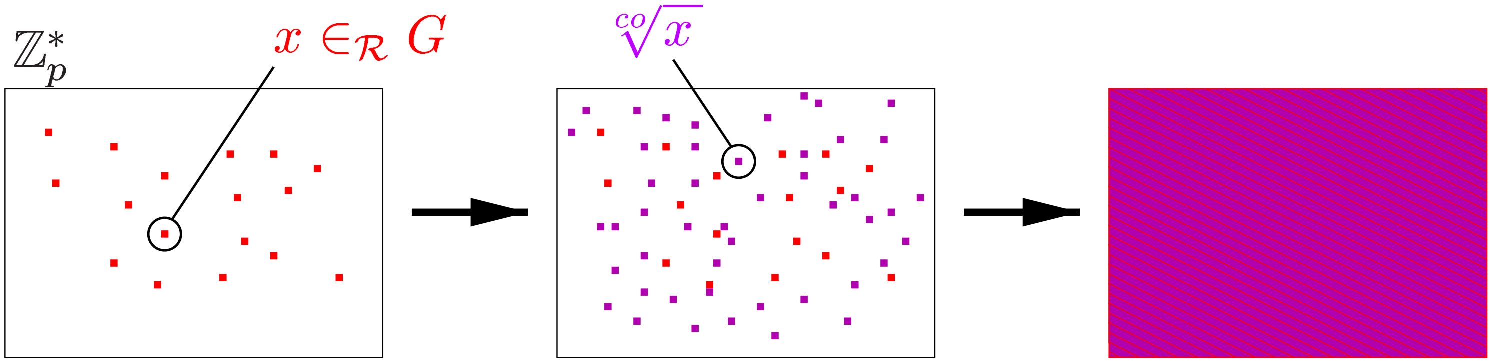
$h^x \in$ subgroup G of $\mathbb{Z}_p^*$ with prime order q

- More efficient
- Security based solely on Diffie-Hellman Decision problem ...
- ... but vulnerable to dictionary attack with "straightforward encryption":

$$\mathrm{Test}(guess, enc) := (\mathrm{E}_{guess}^{-1}(enc))^q \stackrel{?}{\neq} 1 \pmod p$$

New Encryption For Elements Of G

Spread elements of G uniformly over $\mathbb{Z}_p^*$ before encryption



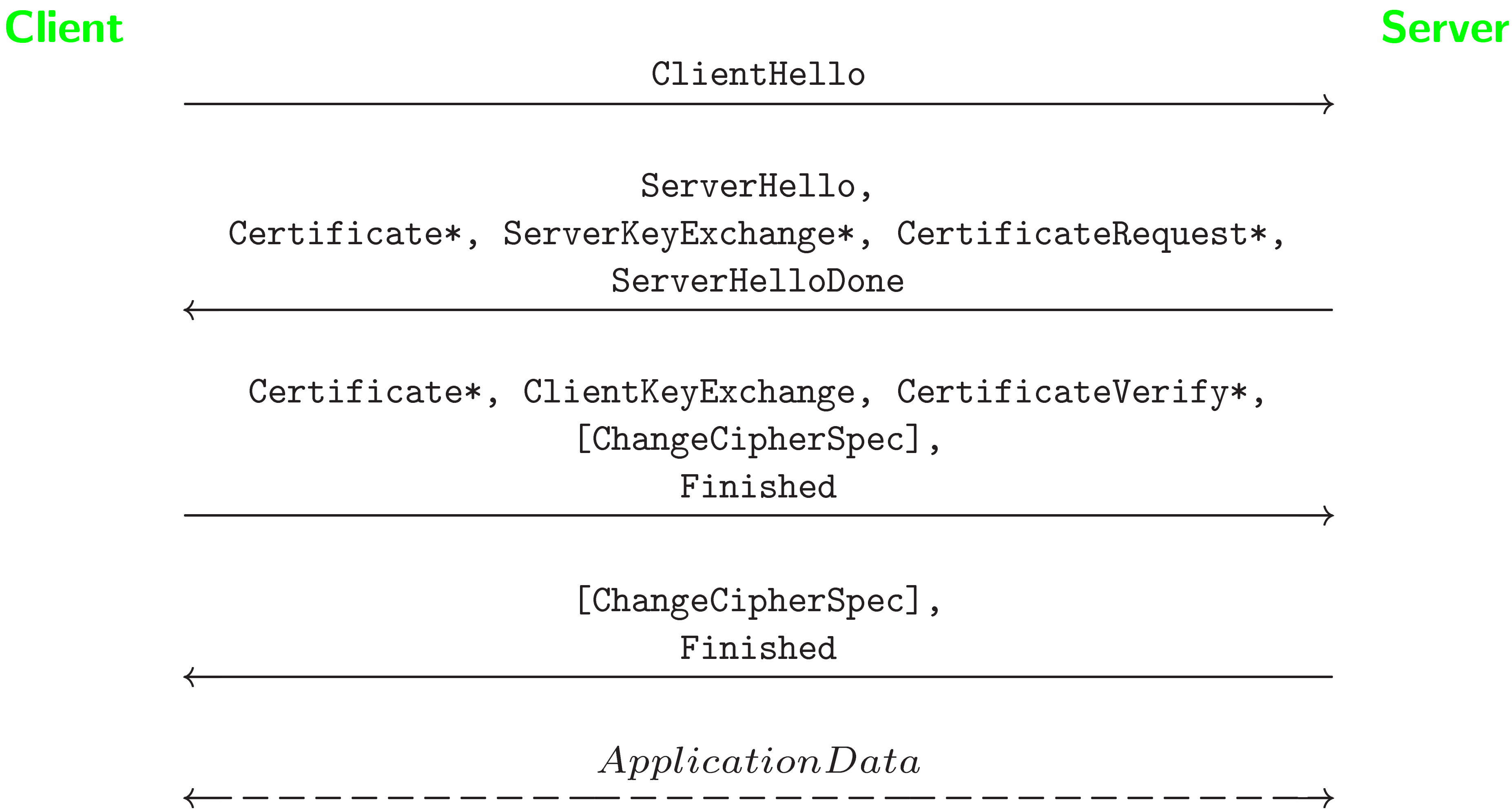
$$\mathrm{E}_G(x) \quad := \quad root \overset{\mathcal{R}}{\leftarrow} \sqrt[co]{x} \; ; \; \mathrm{E}_{\mathbb{Z}_p^*}(root) \qquad [co := (\phi(p)/q)]$$

$$\mathrm{E}_G^{-1}(x) \quad := \quad (\mathrm{E}_{\mathbb{Z}_p^*}^{-1}(r))^{co}$$

Efficiency

- No need to calculate root: choose $g^{x'} \in_{\mathcal{R}} \mathbb{Z}_p^* \;\Rightarrow\; x = x' * co$
- Exponentiation in decryption combined efficiently with other exponentiations.

Overview TLS (RFC 2246)



Adding A New Ciphersuite

Hard constraints

- Don't touch `ClientHello` (backwards compatibility)
- Don't weaken protocol security

Soft constraints

- Limit modifications to `ServerKeyExchange` and `ClientKeyExchange`
- Minimize number of flows
- Minimize changes to state machine
- Reuse existing building blocks

TLS/DH-EKE

Client (password $\boldsymbol{pwd}$)		**Server** (password $\boldsymbol{pwd}$)
	`ClientHello` $\longrightarrow$	
		$x \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q$
	`ServerHello`, `ServerKeyExchange(`h^x`)`, `ServerHelloDone` $\longleftarrow$	
$y \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q$		
	`ClientKeyExchange(`$E_{\boldsymbol{pwd}}(h^y)$`)` $\longrightarrow$	
$k_{mstr} \leftarrow (h^x)^y$ $k_{conf} \leftarrow \mathcal{G}_1(k_{mstr})$ $k_{sess} \leftarrow \mathcal{G}_2(k_{mstr})$		$k_{mstr} \leftarrow (h^y)^x$ $k_{conf} \leftarrow \mathcal{G}_1(k_{mstr})$ $k_{sess} \leftarrow \mathcal{G}_2(k_{mstr})$
	`[ChangeCipherSpec]`, `Finished(`$\mathrm{MAC}(k_{conf}, \mathrm{prev.\ msgs.})$`)` $\longleftarrow$	
accept if OK(`Finished`)		
	`[ChangeCipherSpec]`, `Finished(`$\mathrm{MAC}(k_{conf}, \mathrm{prev.\ msgs.})$`)` $\longrightarrow$	
		accept if OK(`Finished`)
	$\longleftarrow$ *ApplicationData* $\longrightarrow$	

Rejected Alternatives To DH-EKE

SPEKE

+ No encryption with passwords
+ Extends easily to elliptic curves
− More flows or change in `Finished`

SRB

+ No encryption with passwords bb
+ Lowest computation costs
− More flows or change in `Finished`

Server Public Keys

+ Simple protocols
+ Fully proven in formal model
− More infrastructure needed
− Risks in certificate management

Conclusions

Password-based protocols

- can be made secure
- tolerate "clueless" users
- are minimal in infrastructure requirements

⇒ Highly useful in many circumstances

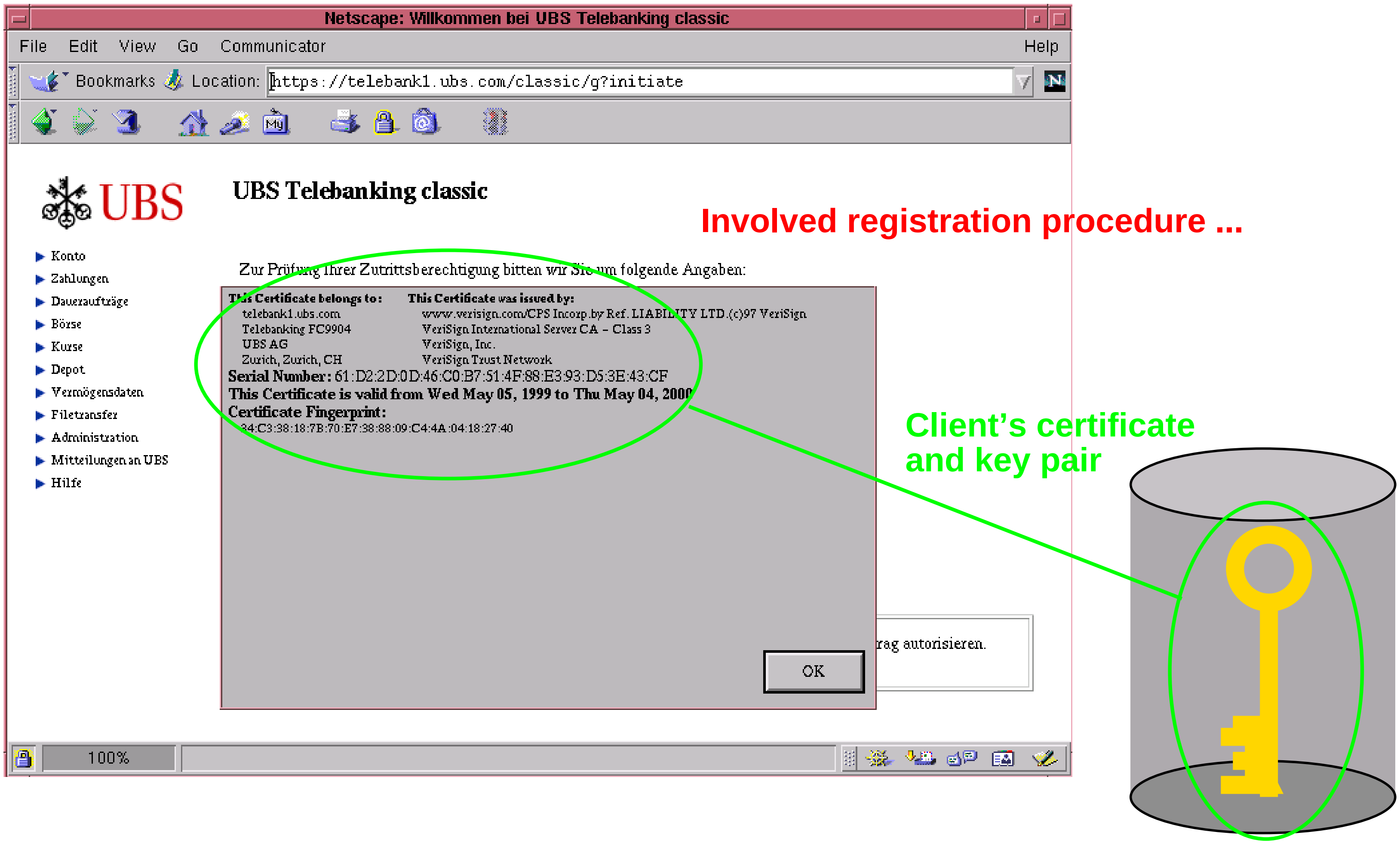
Our integration of DH-EKE in TLS ...

- ideally complements existing ciphersuite
- is as non-intrusive as possible
- requires the minimal number of flows
- has competitive performance

⇒ Let's add DH-EKE to the TLS standard ... :-)

... but patents must be resolved first ... :-(

SSL With Client Certification . . .



Password-encrypted key on harddisk?

Smartcard vs. “Dumbcard”

Smartcard for PK-based key-exchange

- Personal
- Authentication of card and user
- Storage of keys and certificates
- Tamper-resistance (secrecy of key)
- Trusted I/O to user
- Math co-processor and secure random number generator

“Dumbcard” for password-based key-exchange

- Impersonal
- Authentication of card
- Tamper-evidence (integrity of card)
- Trusted I/O to user
- Math co-processor and secure random number generator