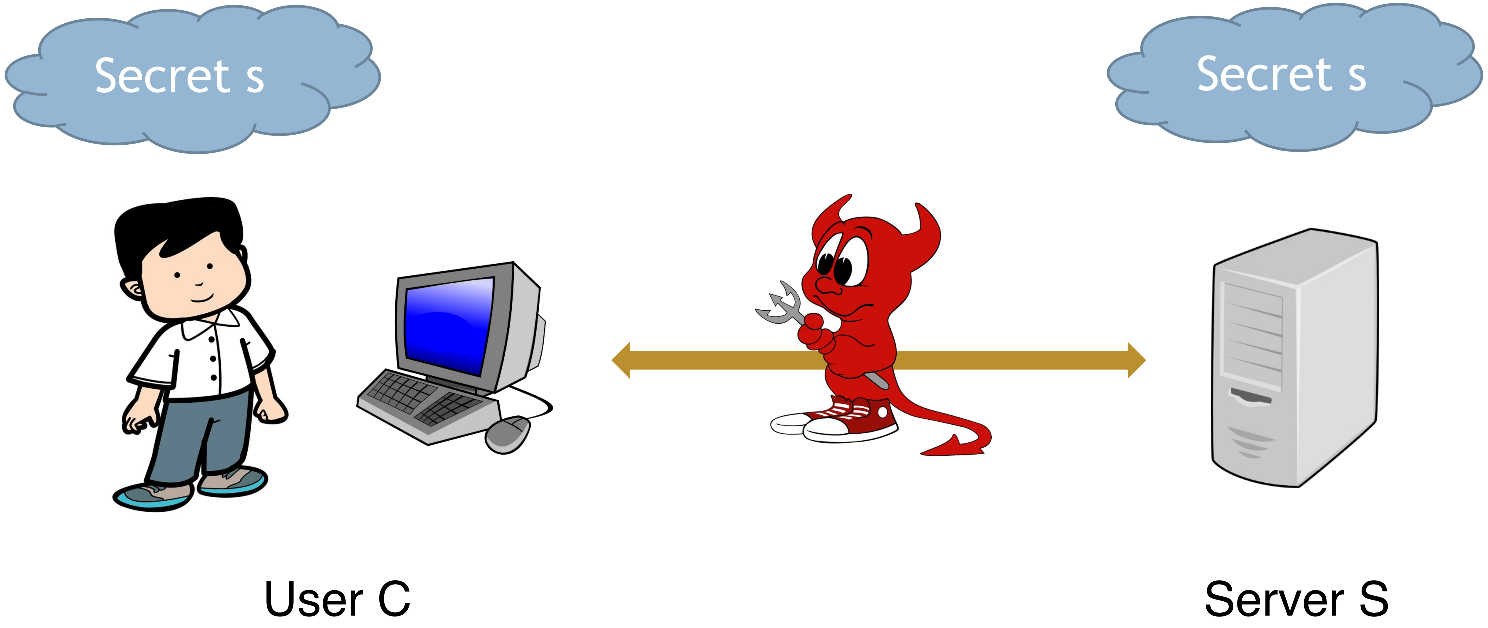
Human Computing for Handling Strong Corruptions in Authenticated Key Exchange

Alexandra Boldyreva **Shan Chen** Pierre-Alain Dupont David Pointcheval

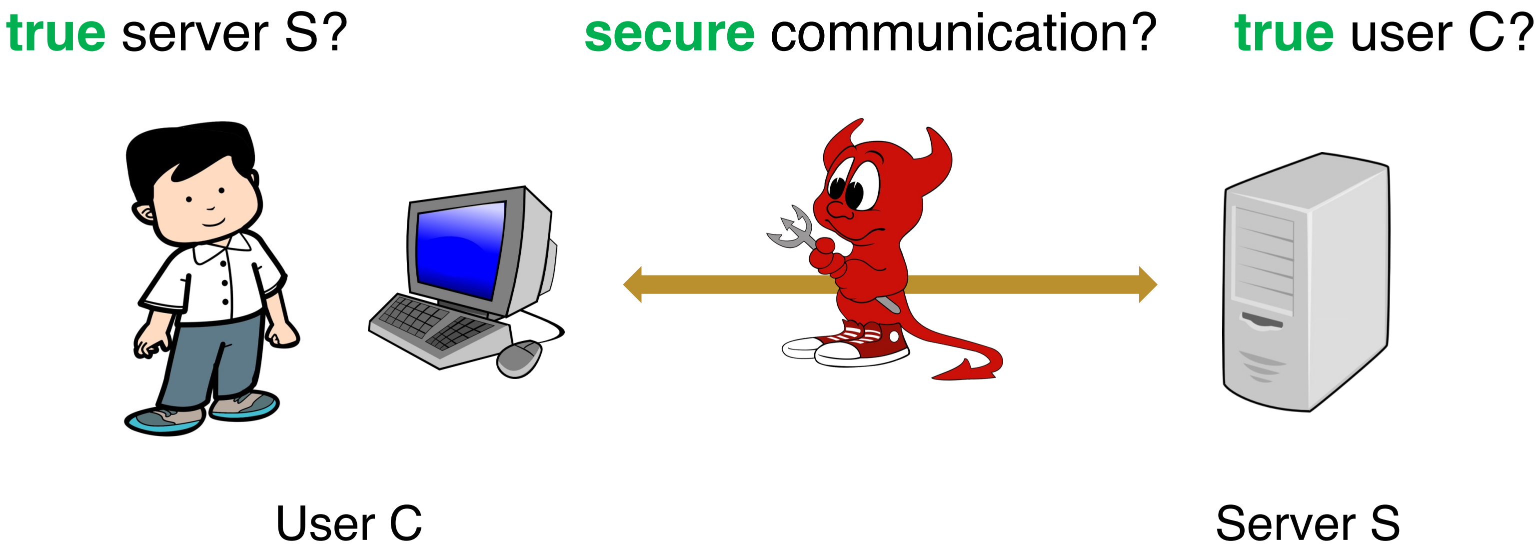
Georgia Institute of Technology École Normale Supérieure

Background



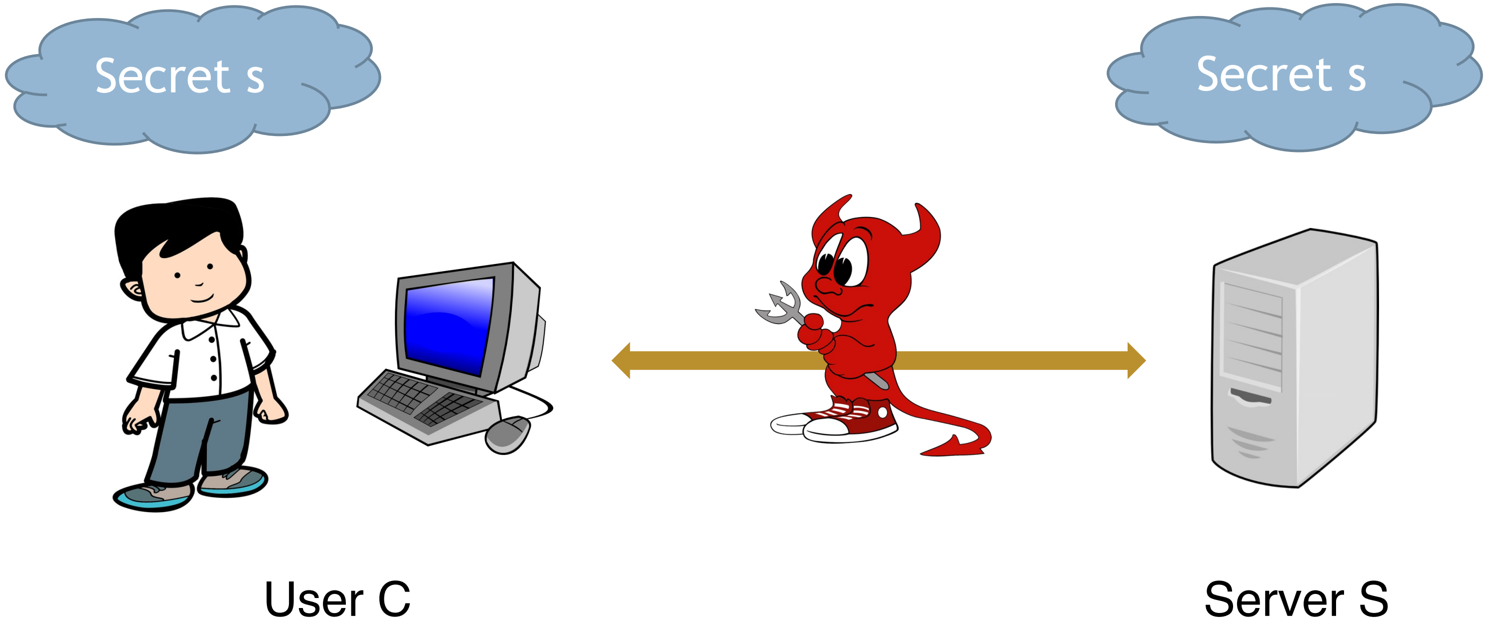
- Example: Log in to your Facebook account…

Background



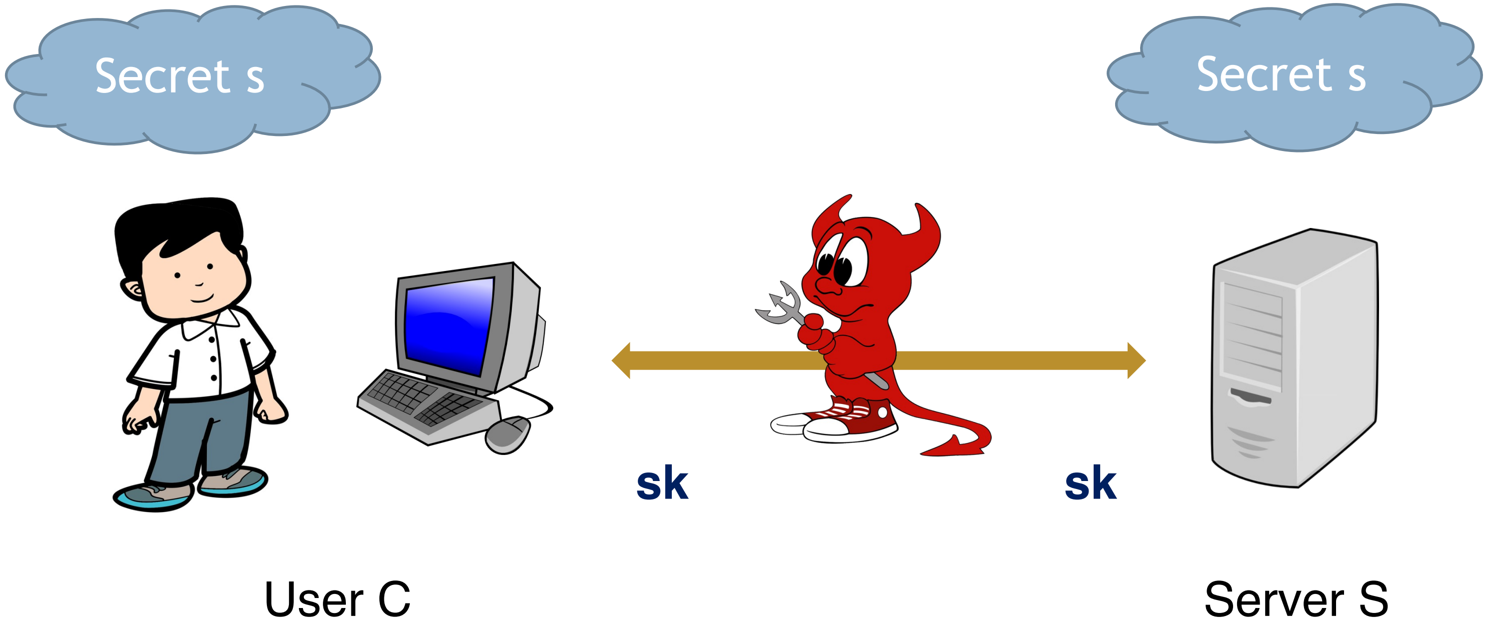
- Example: Log in to your Facebook account…

Background



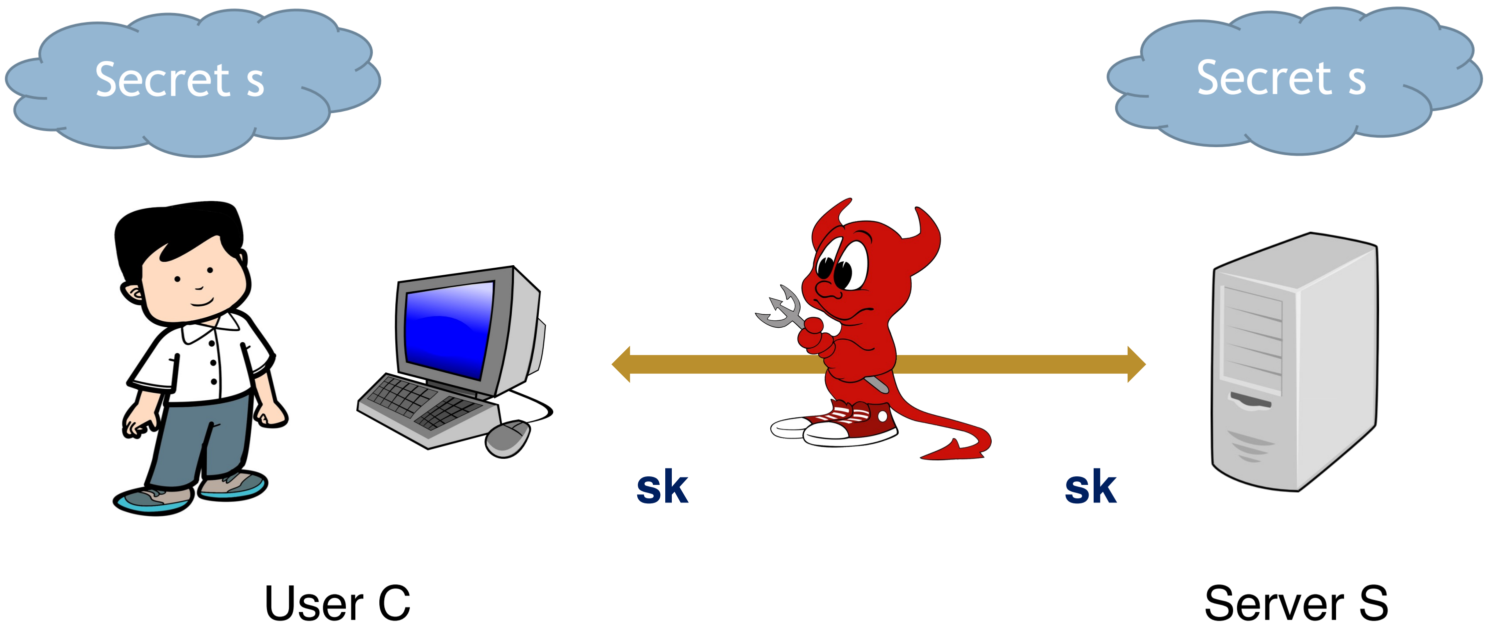
- Solution? **Authenticated Key Exchange**

Background



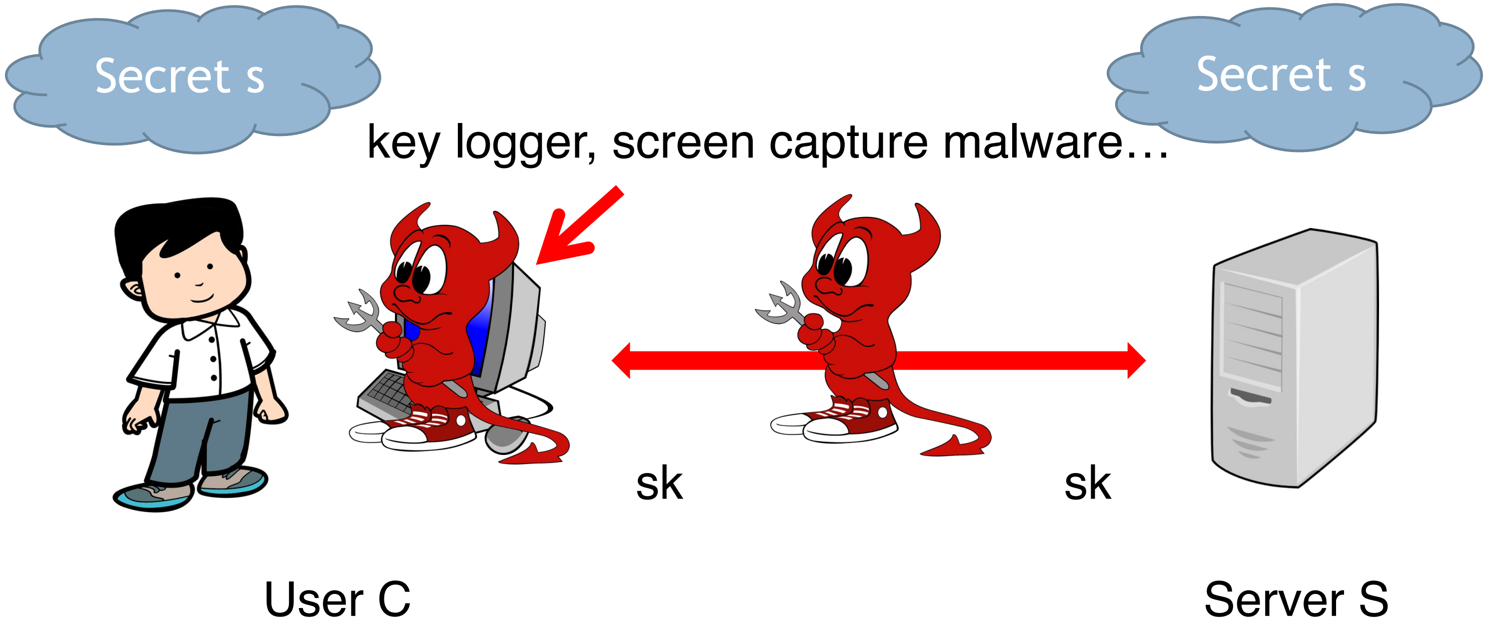
- Solution? **Authenticated Key Exchange**
  - **Session key**: protect the communication & authenticate the involved parties

Background



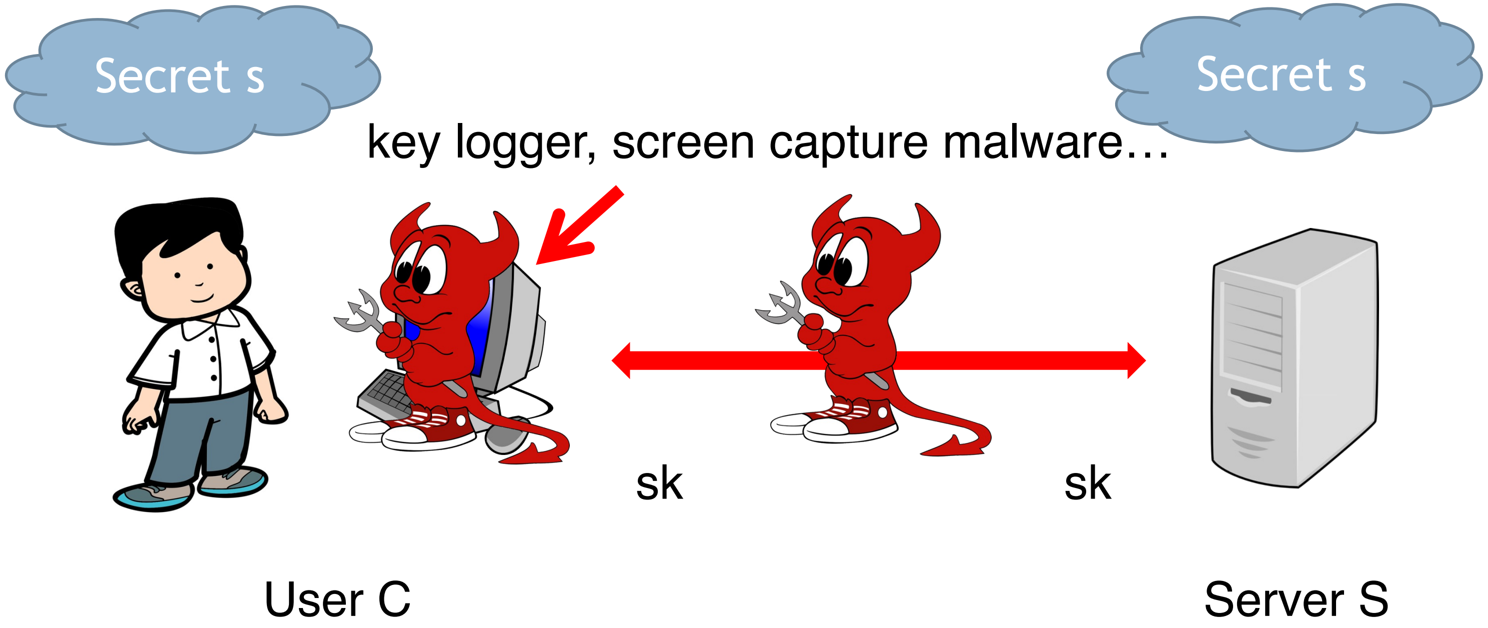
- Solution? **Authenticated Key Exchange**
  - Protect against session key compromise (**weak corruptions**).

Motivation



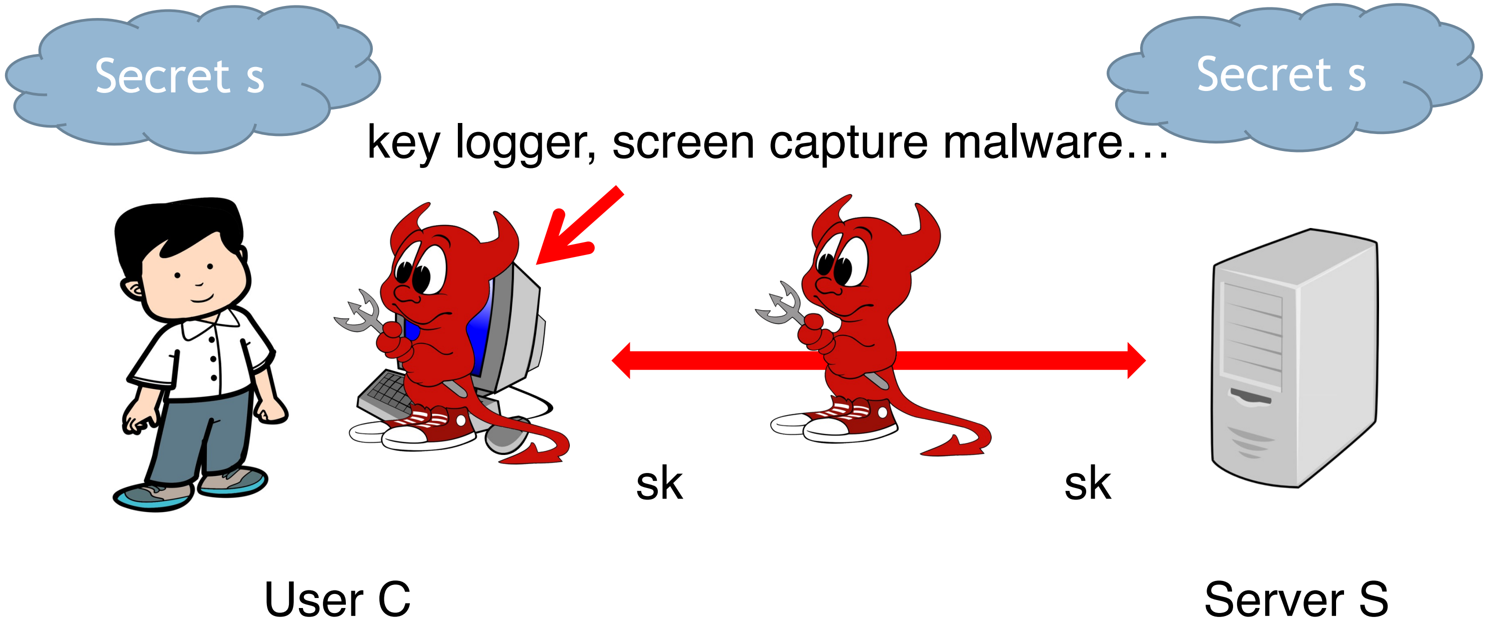
- What if the terminal has been **compromised**? (**strong corruptions**)
  - Happens in real life, sometimes the terminal may be fully controlled.

Motivation



- What if the terminal has been **compromised**? (**strong corruptions**)
  - Some existing protocols can protect the **past** sessions (**forward secrecy**).

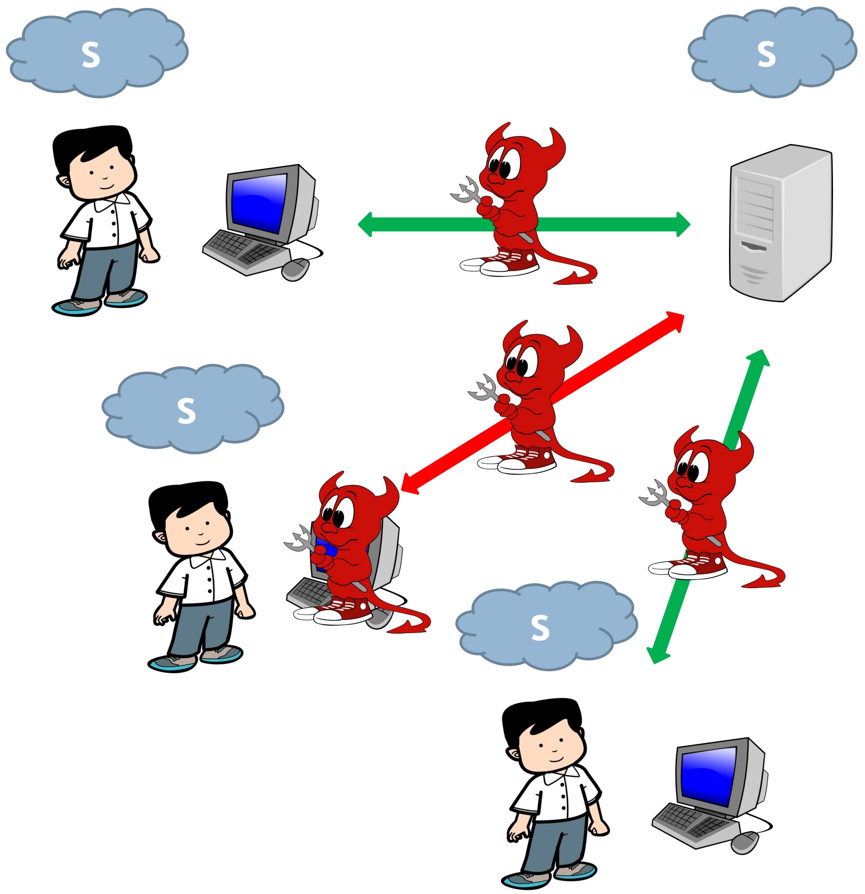
Motivation



- What if the terminal has been **compromised**? (**strong corruptions**)
  - **No solution** for protecting **future** sessions (because s is leaked)!

Our Goal

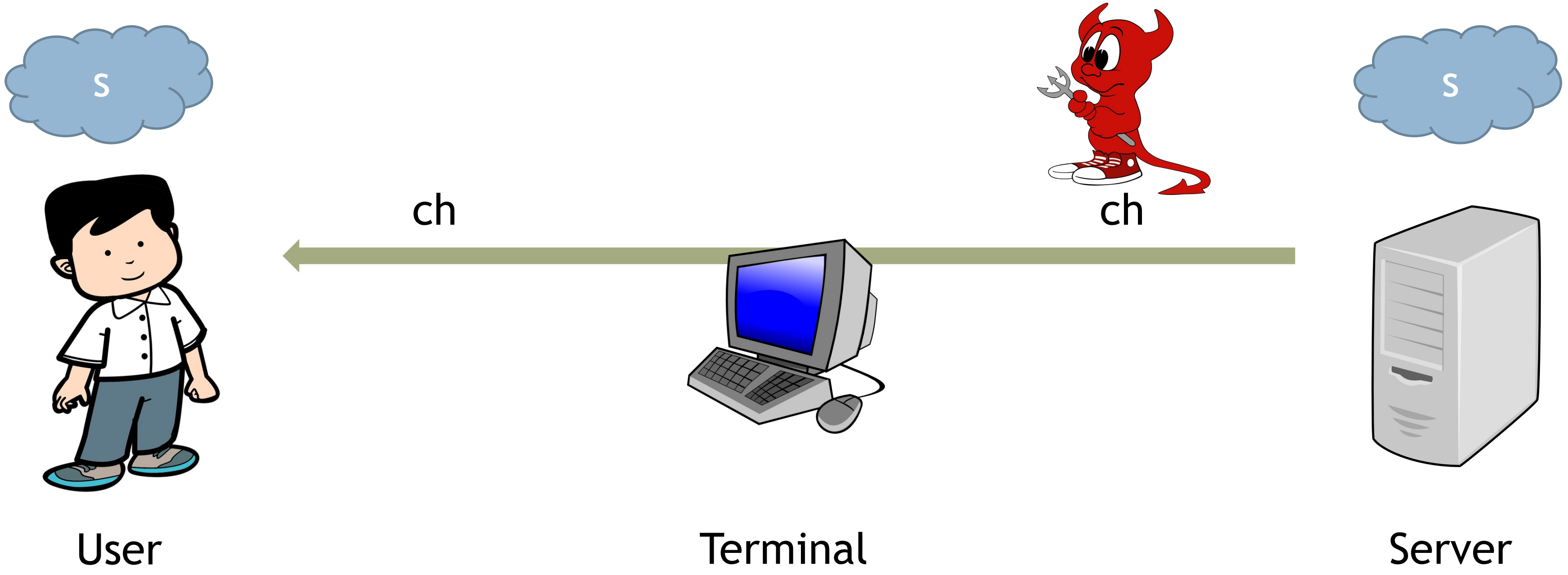
- Given **compromised** terminals, a user's **past** and **future** sessions from other **secure** terminals are still protected, even though the **same** long-term secret s is used.



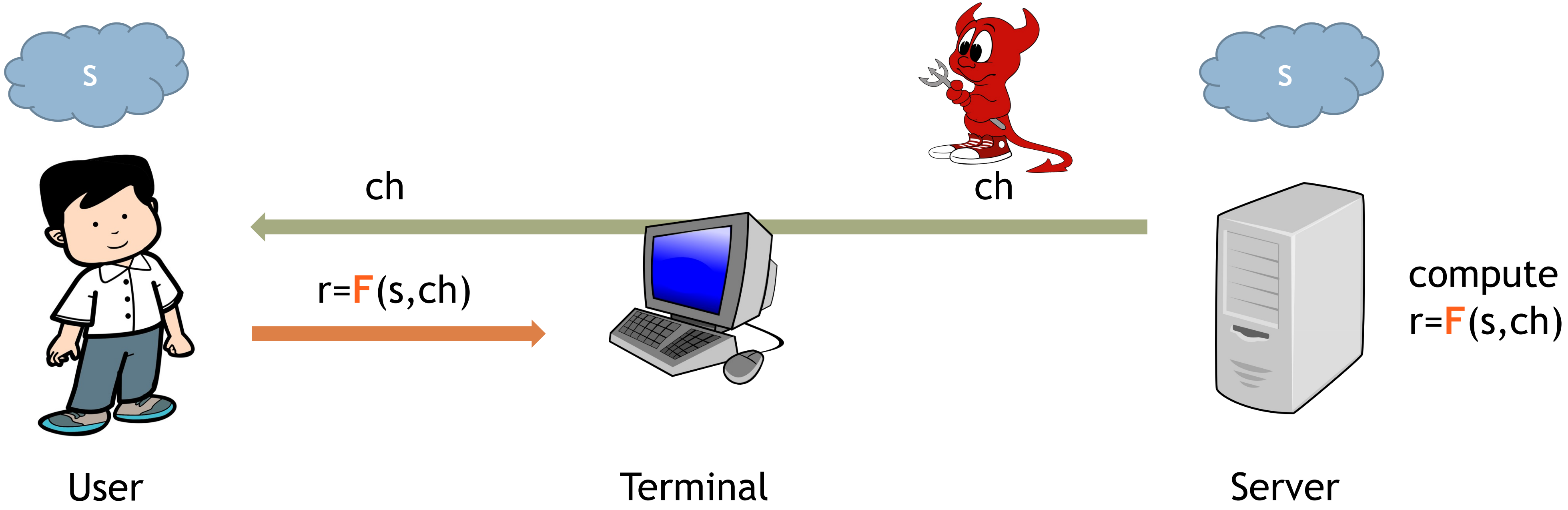
Basic Idea

- Can not enter long-term secret s into the terminal.
  - Use a **challenge-response function** instead!

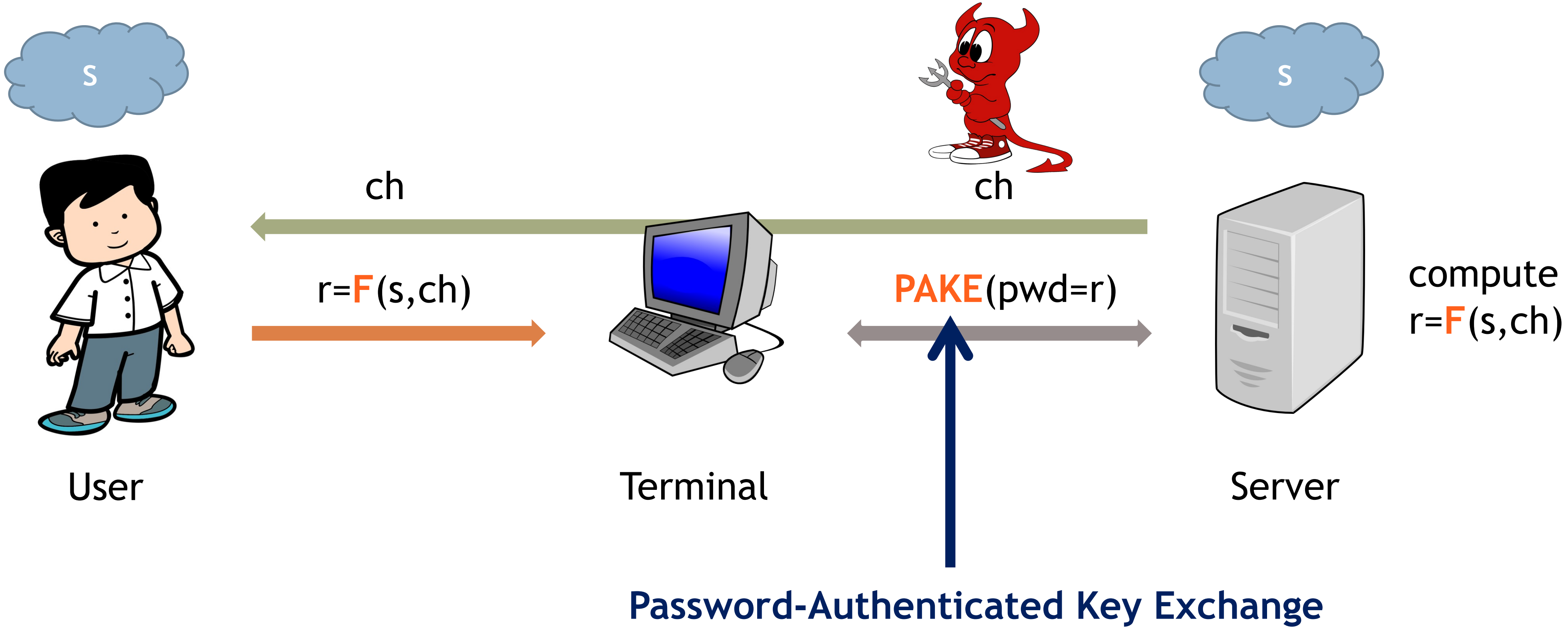
Basic Idea



Basic Idea

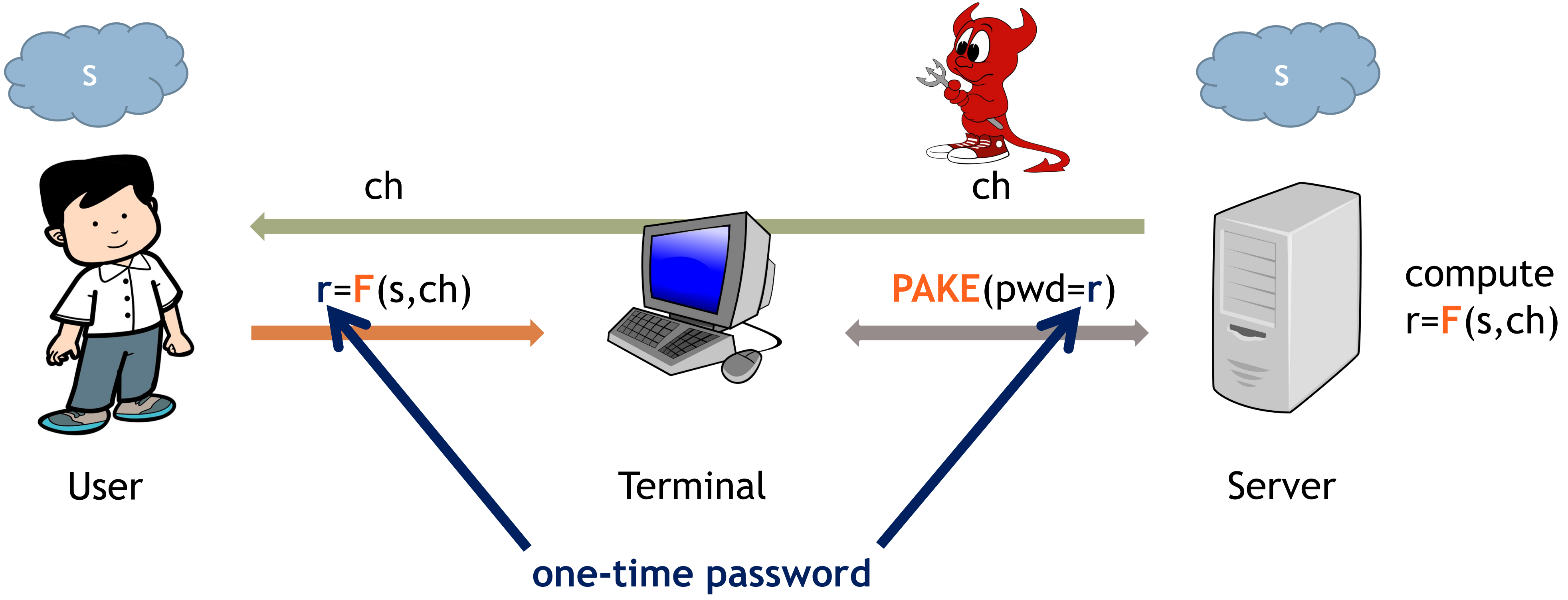


Basic Idea

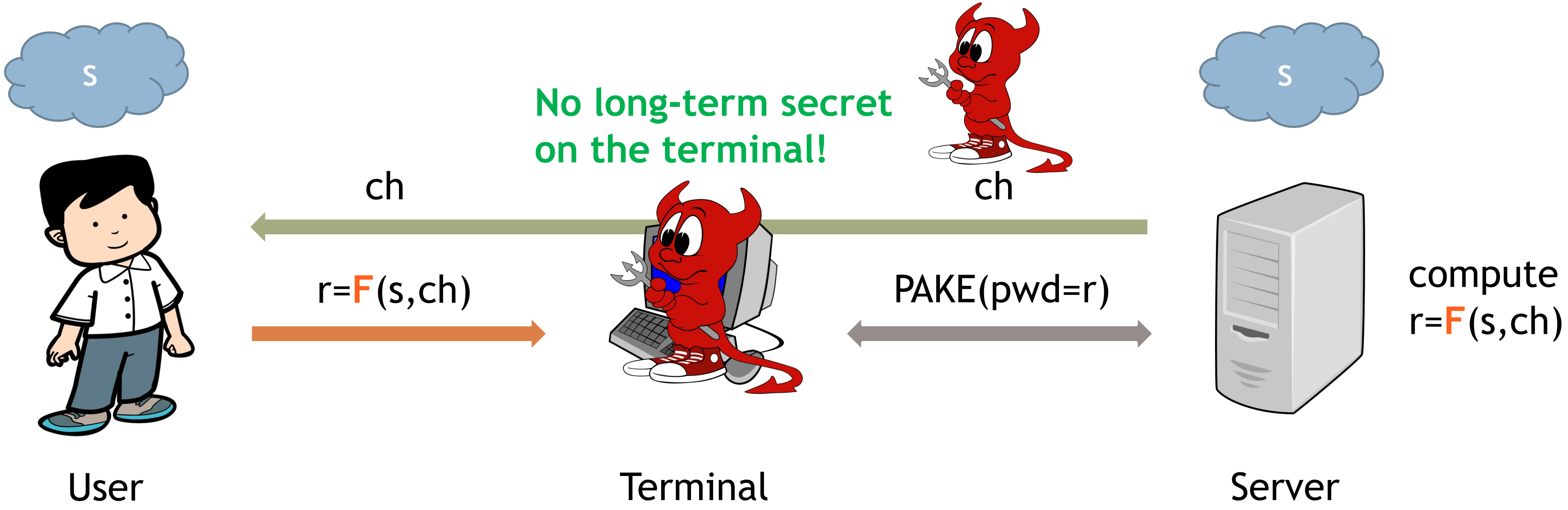


s
s
ch
ch
r=F(s,ch)
PAKE(pwd=r)
compute
r=F(s,ch)
User
Terminal
Server
Password-Authenticated Key Exchange

Basic Idea

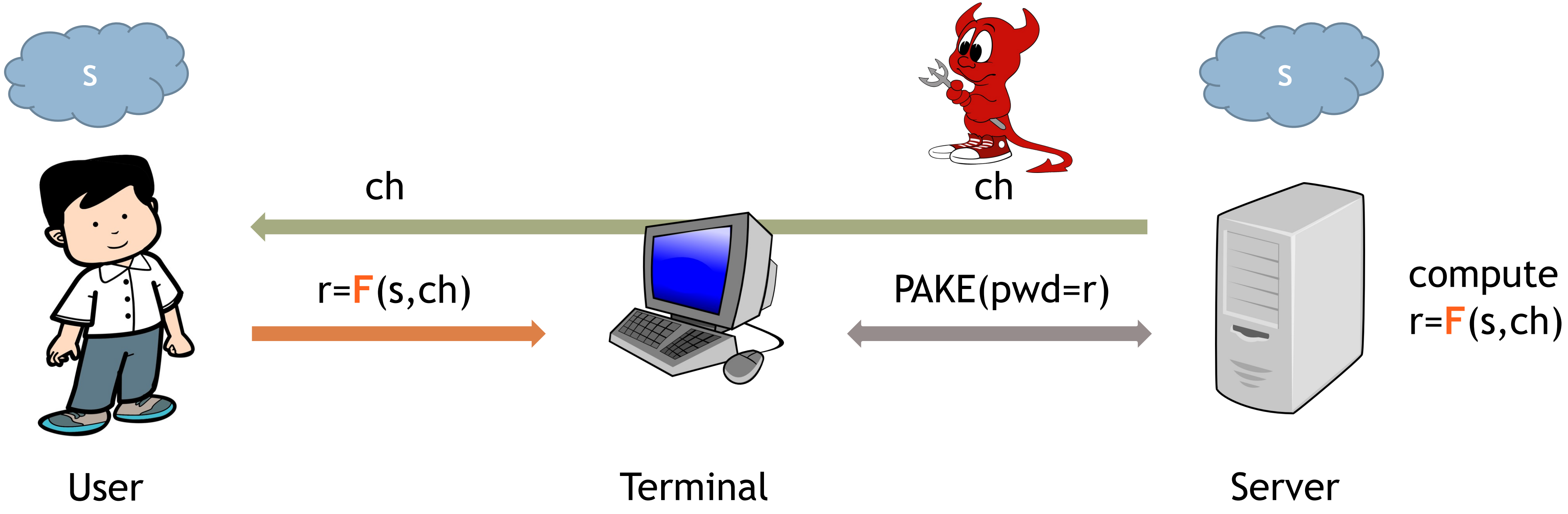

s
s
ch
ch
r=F(s,ch)
PAKE(pwd=r)
compute
r=F(s,ch)
User
Terminal
Server
one-time password

Basic Idea



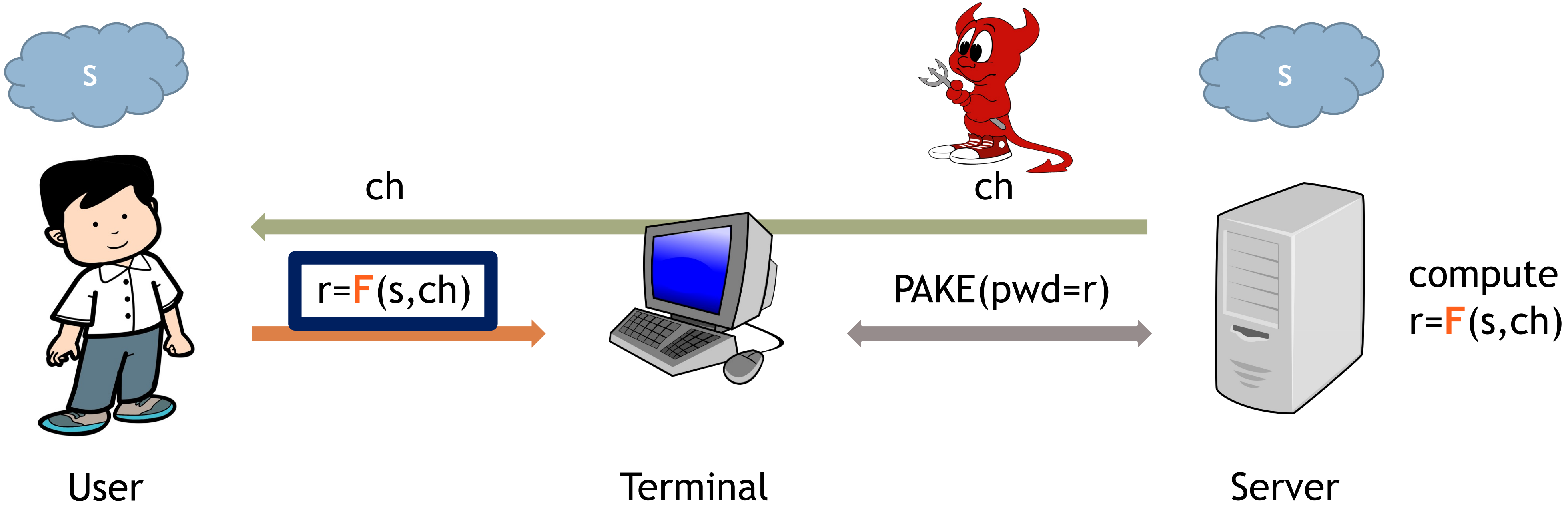
- Long-term secret is **never** typed in or stored on the terminal.
  - Only the challenge-response pairs (ch,r) can be revealed.

Basic Idea



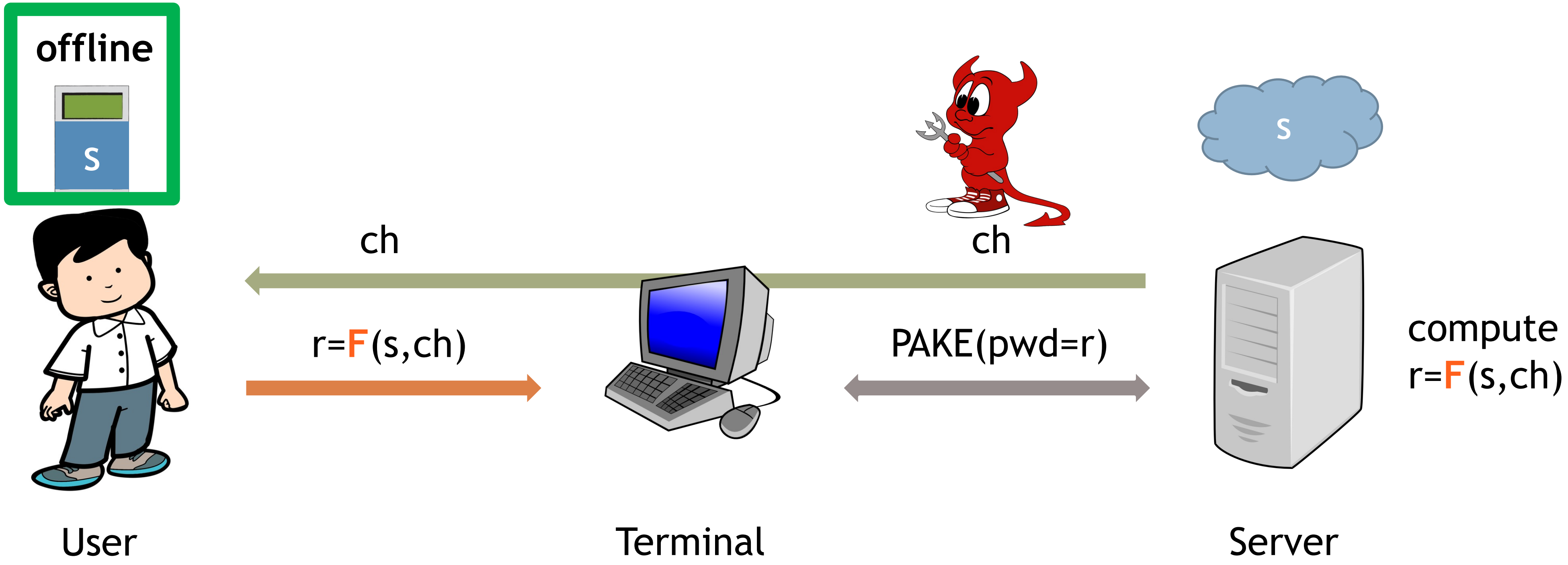
- Looks good, but...
  - there are some unsolved problems.

Basic Idea



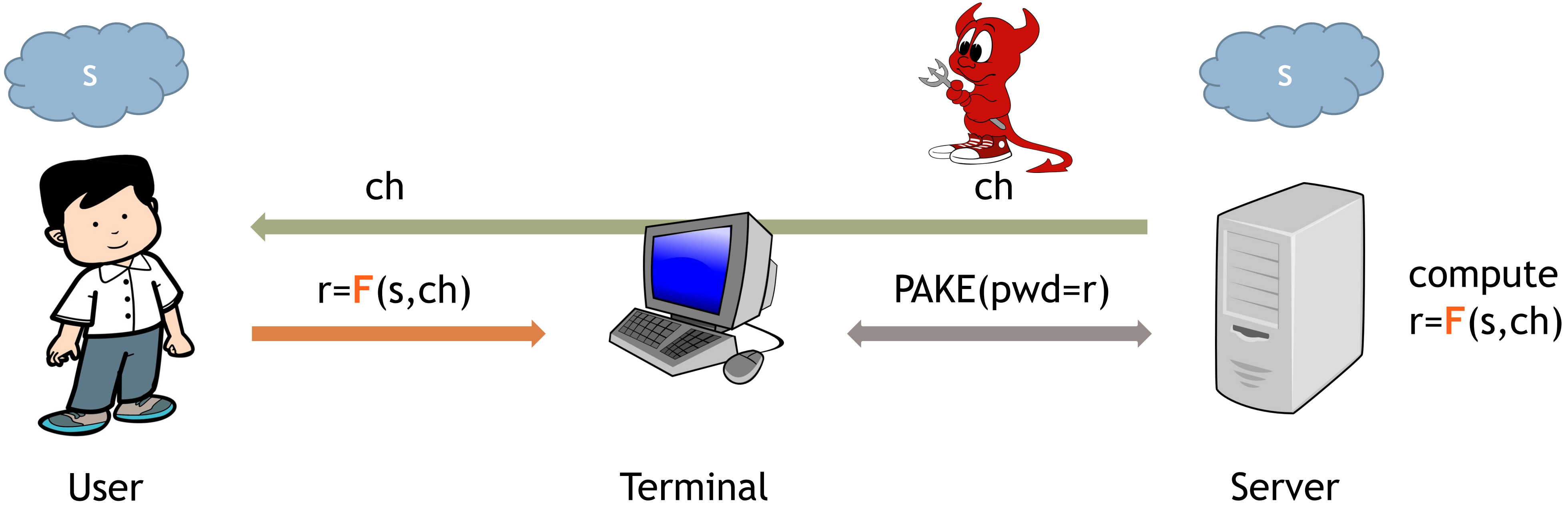
- How to construct F?
  - not trivial: human-computable & secure

Basic Idea with Additional Device



- Second approach: **additional secure device**
  - human user's burden reduced & more practical

Basic Idea

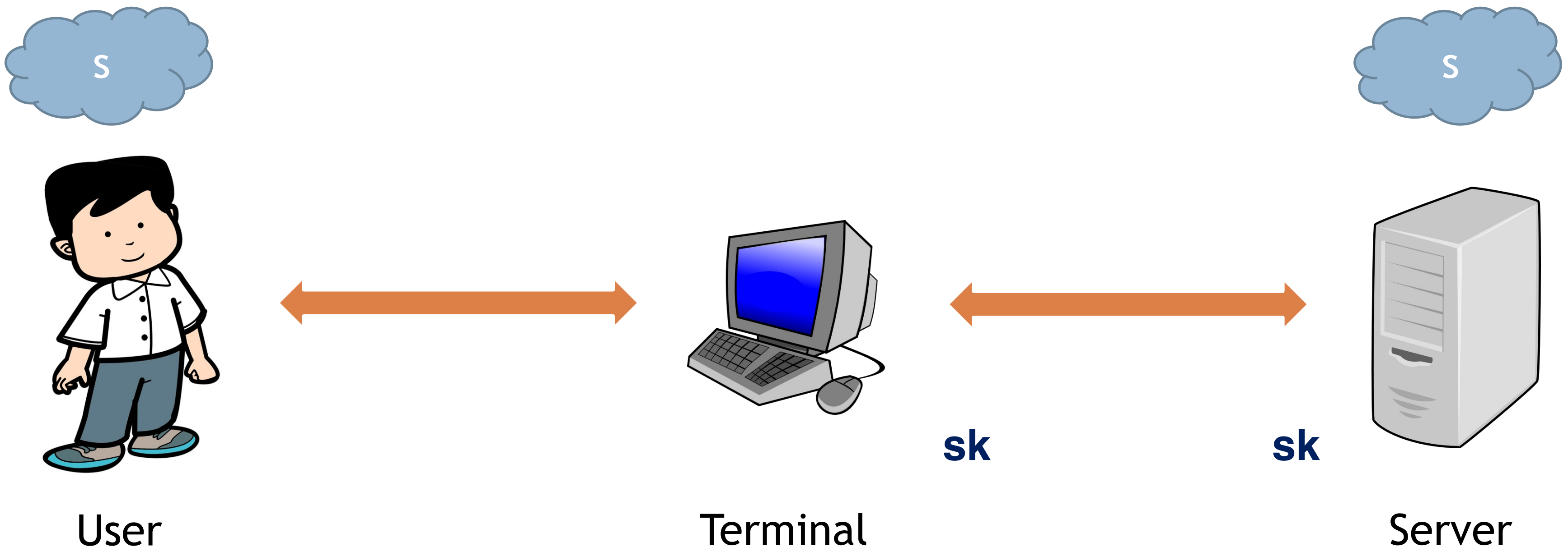


- Can this protocol achieve our goal?

Recall: Provable Security Approach

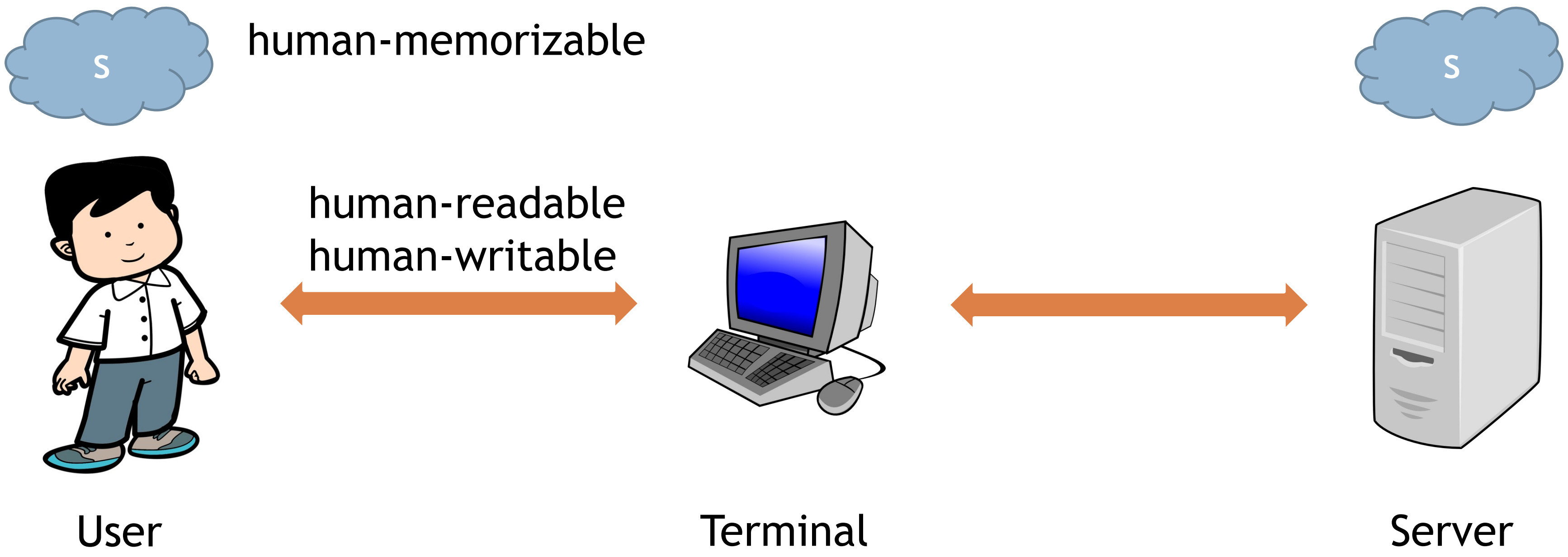
- How to show a protocol is secure?
  - Define the **syntax**:
    - What is a protocol?
  - Define the **security model**:
    - What can the attacker do? What are the security goals?
  - **Prove by reduction** the protocol satisfies the security goals under reasonable hardness assumptions.

HAKE Syntax



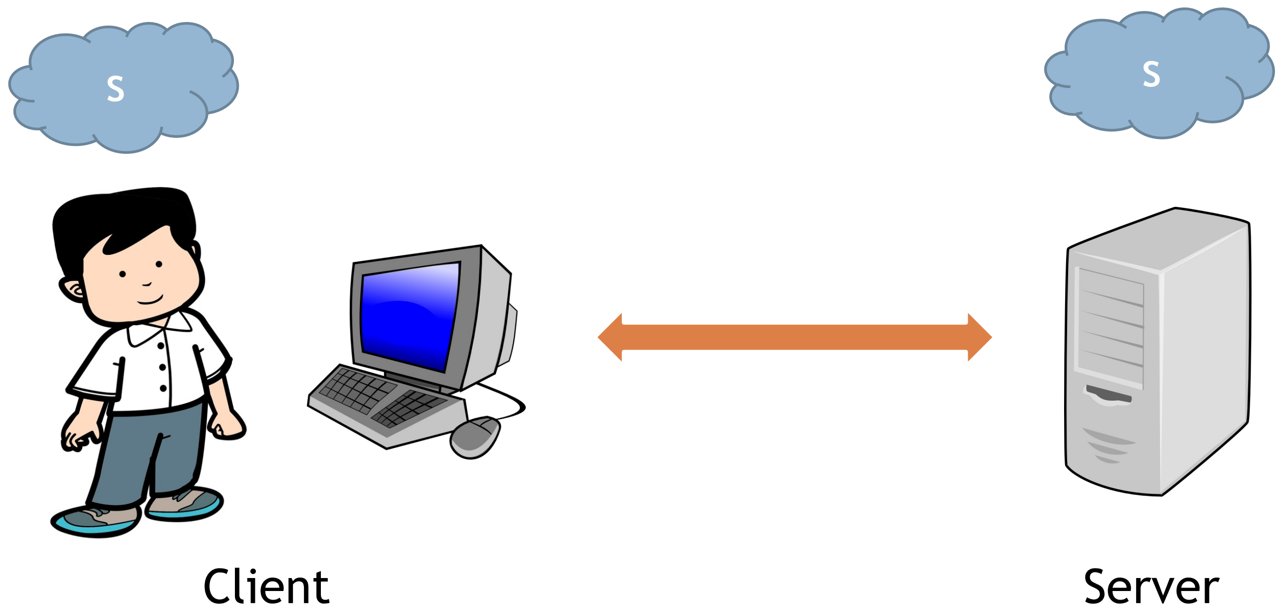
- We define a new protocol called **Human Authenticated Key Exchange (HAKE)** among **3** parties (instead of 2).

HAKE Syntax



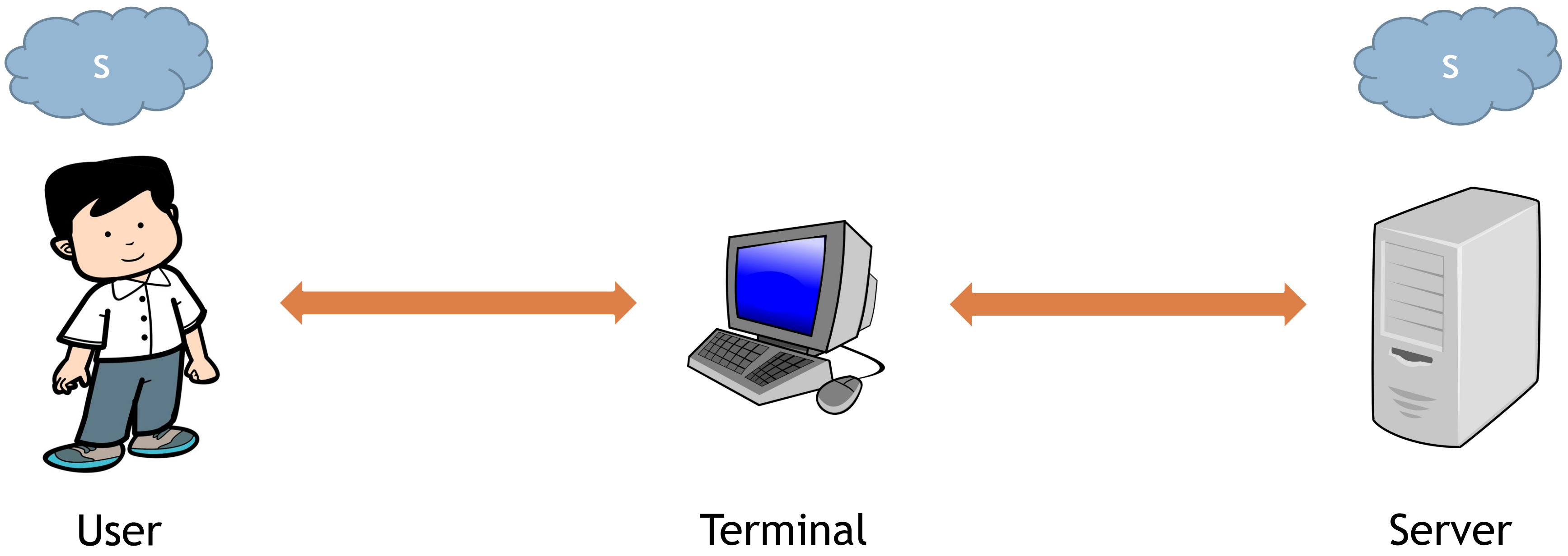
- Human-memorizable: simple enough to be memorized by an average human.
- Human-readable/writable: short sequence of digits, letters, etc.

HAKE Security Model



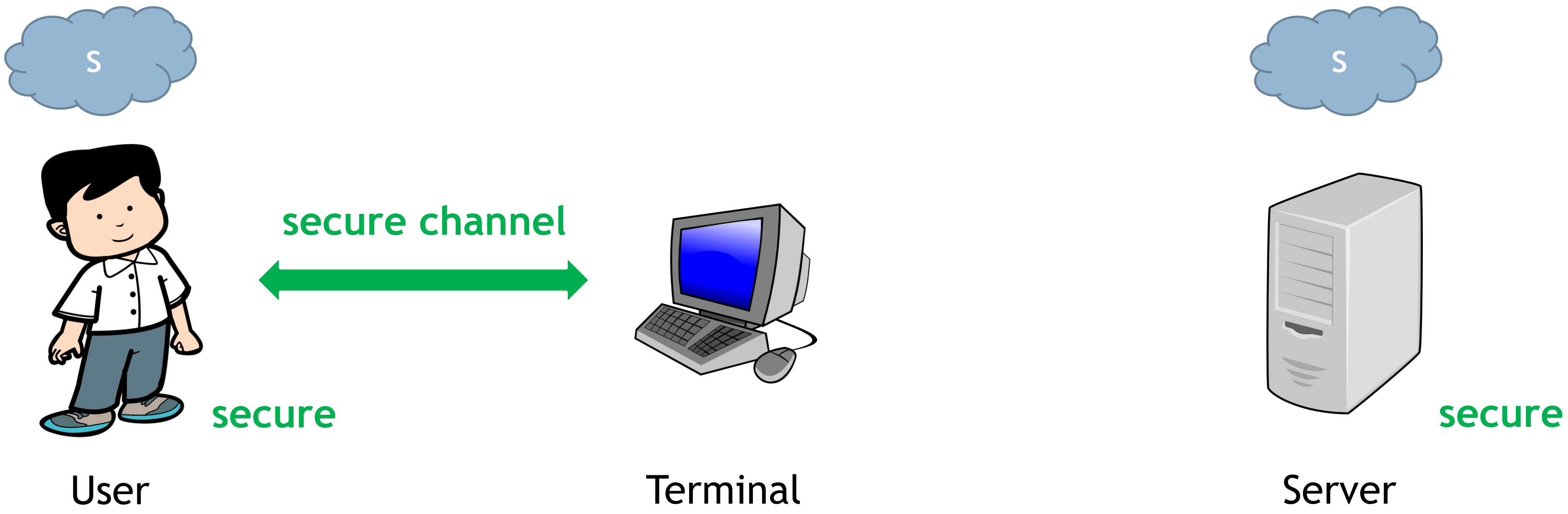
- **Non-trivial** extension of the BPR model [BPR00] for PAKE.

HAKE Security Model



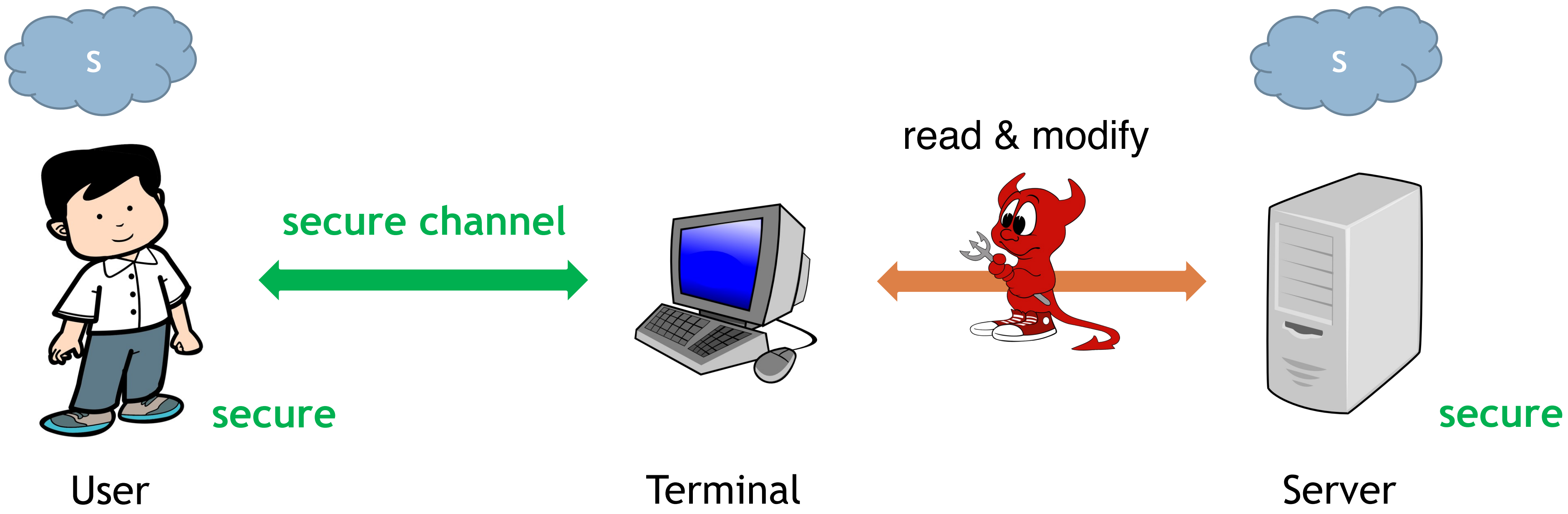
- **Non-trivial** extension of the BPR model [BPR00] for PAKE.
  - human interactions between the user and the terminal

HAKE Security Model



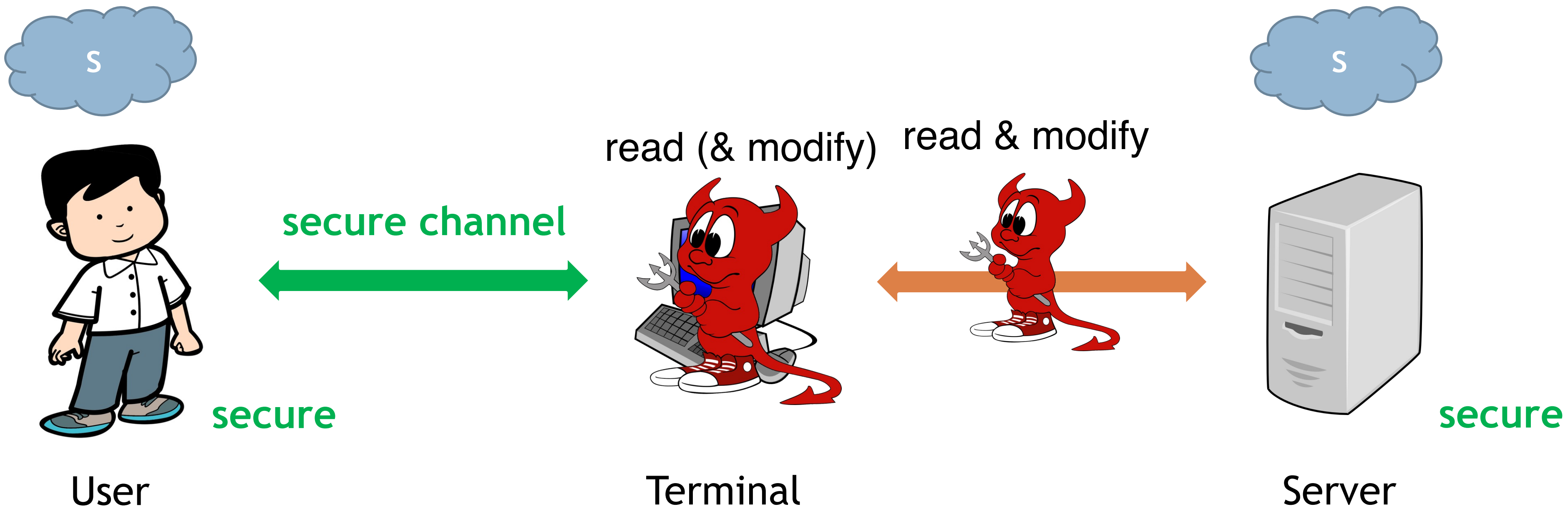
- What can the attacker do?

HAKE Security Model



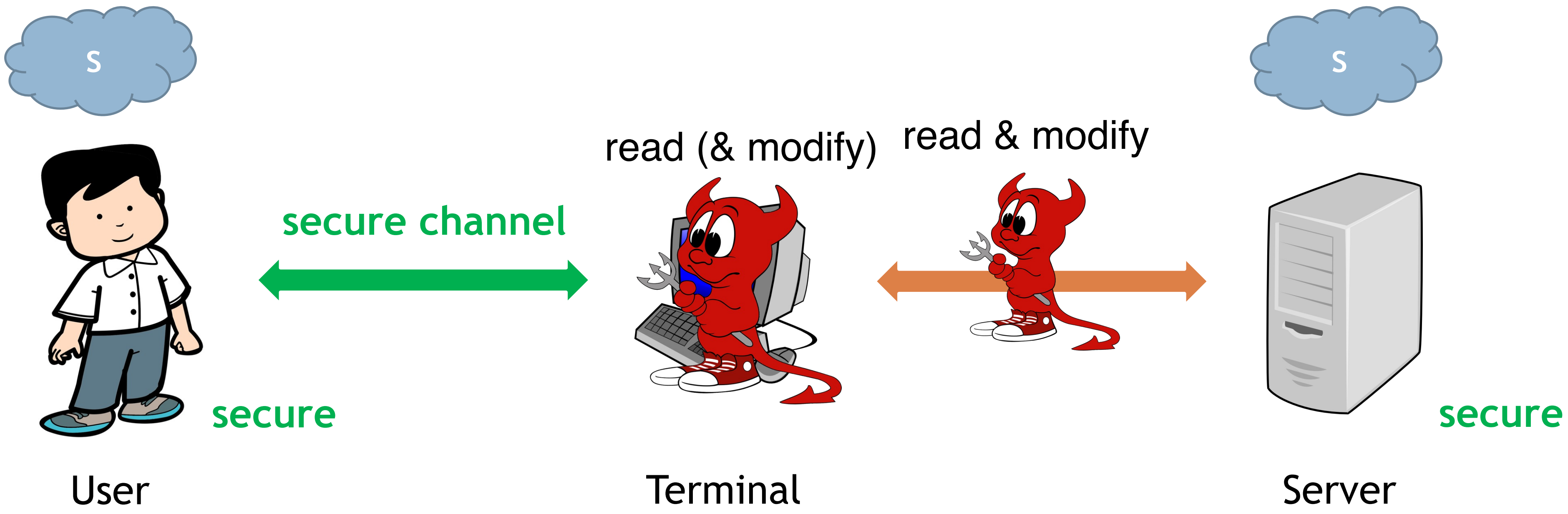
- What can the attacker do?
  - pretend to be the true server/user, guess sk...

HAKE Security Model



- What can the attacker do?
  - corrupt the current session, analyze the user's long-term secret s

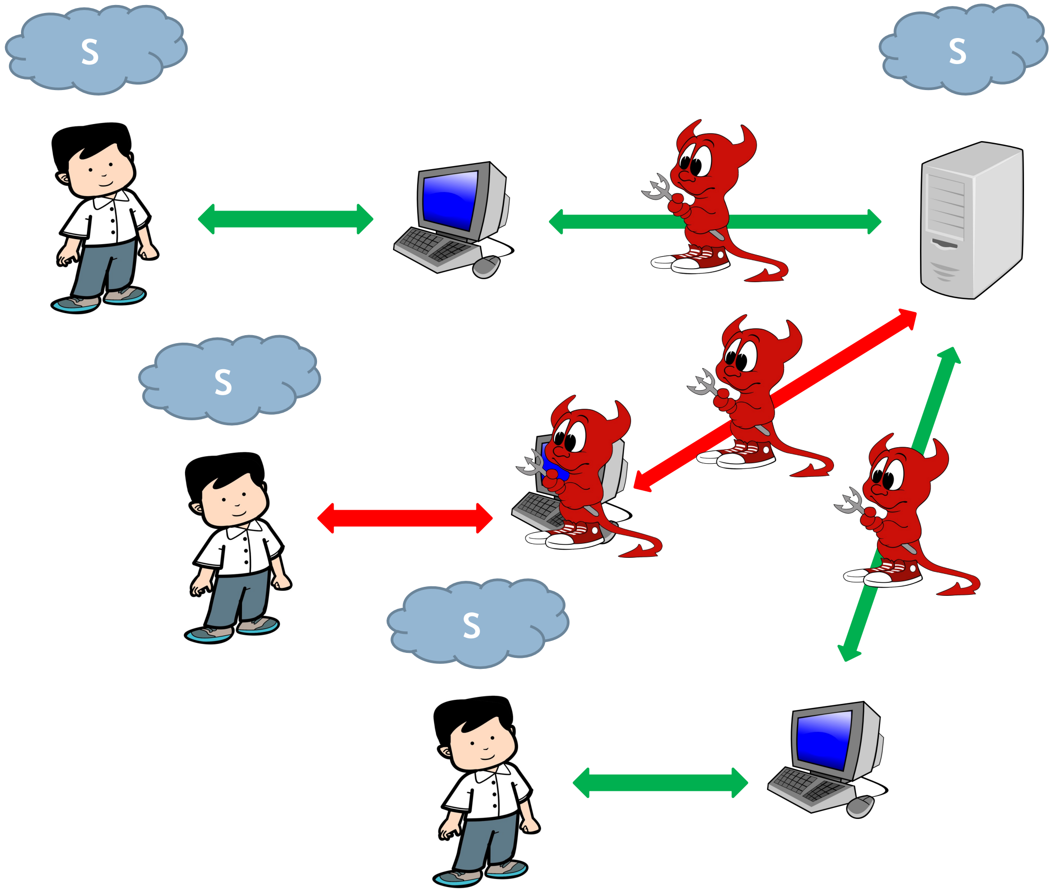
HAKE Security Model



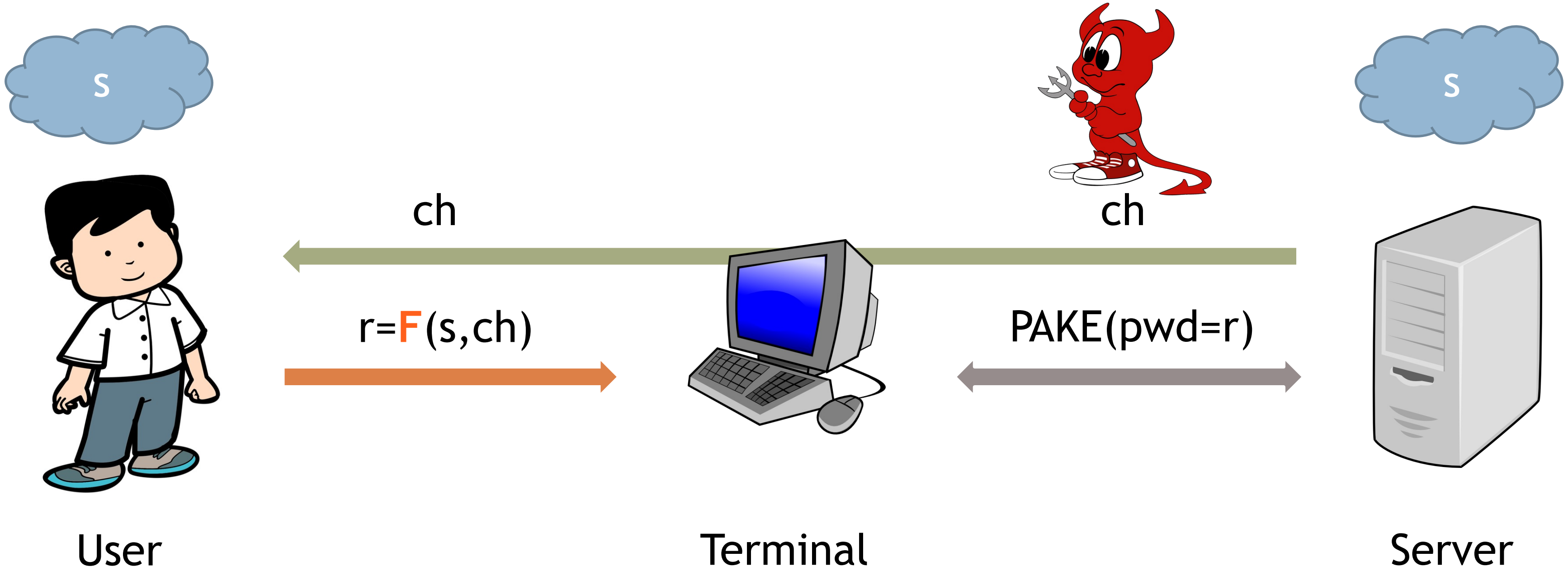
- We model **strong corruptions** for all past and **future** sessions.
  - BPR only deals with past sessions for such active attackers.

HAKE Security Model

- What are the security goals?
  - **privacy** and **authentication** for past and future sessions from other secure terminals (given compromised terminals)
- Terminologies:
  - **Privacy**: no information is leaked about the session key.
  - **Authentication**: each party (user or server) builds a secure session with the right other party.

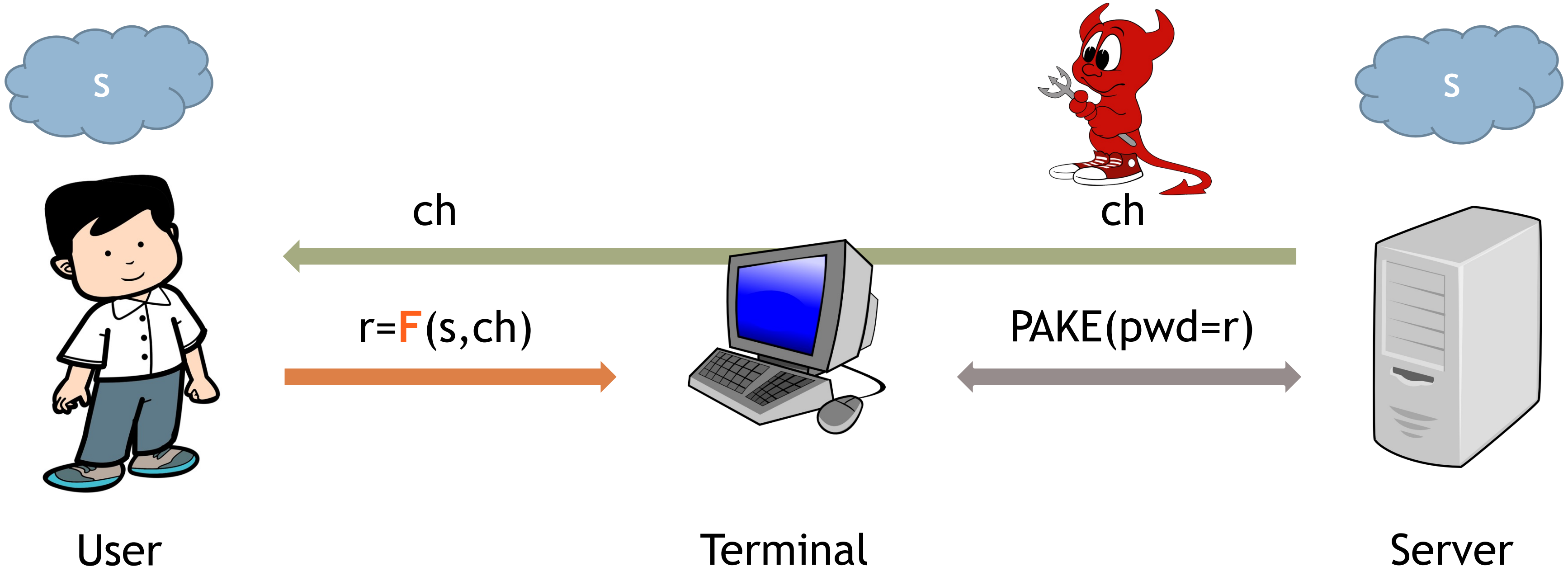


Recall Basic Idea



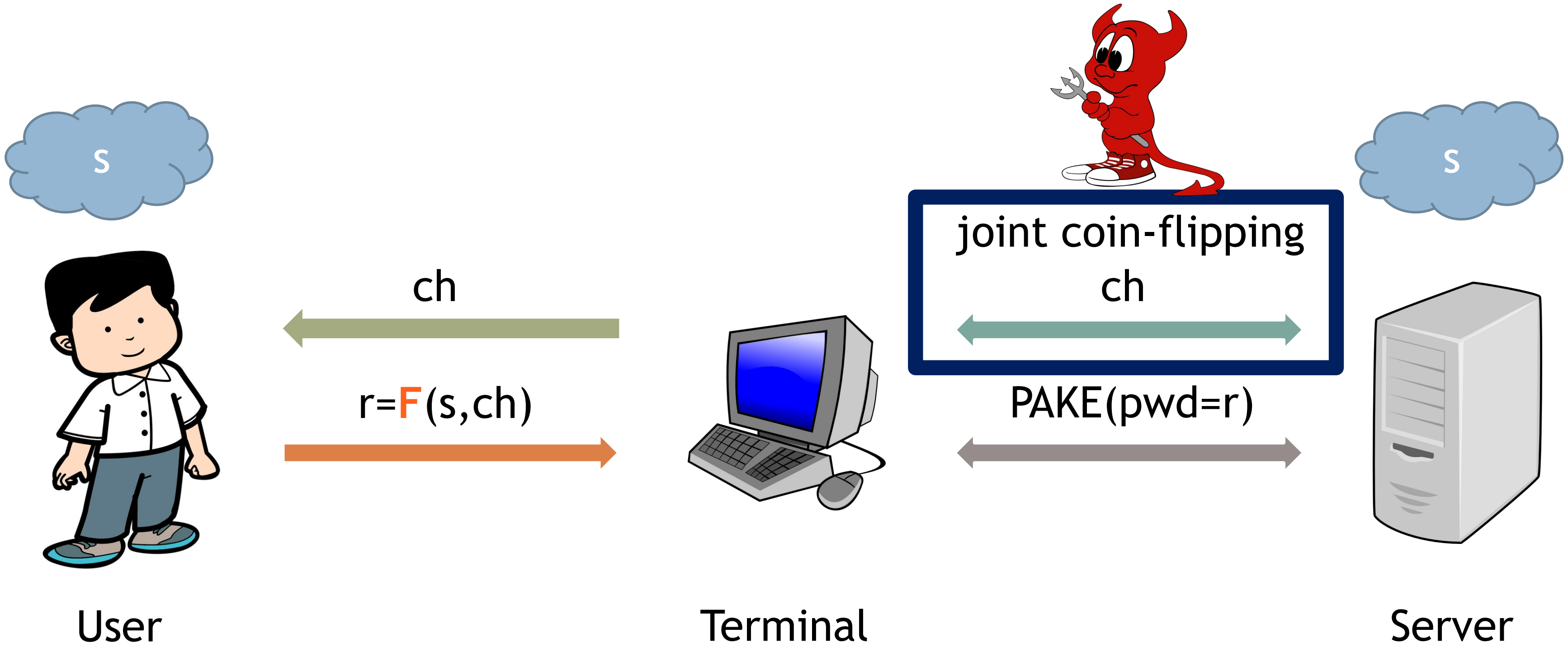
- Is this protocol secure?

Recall Basic Idea



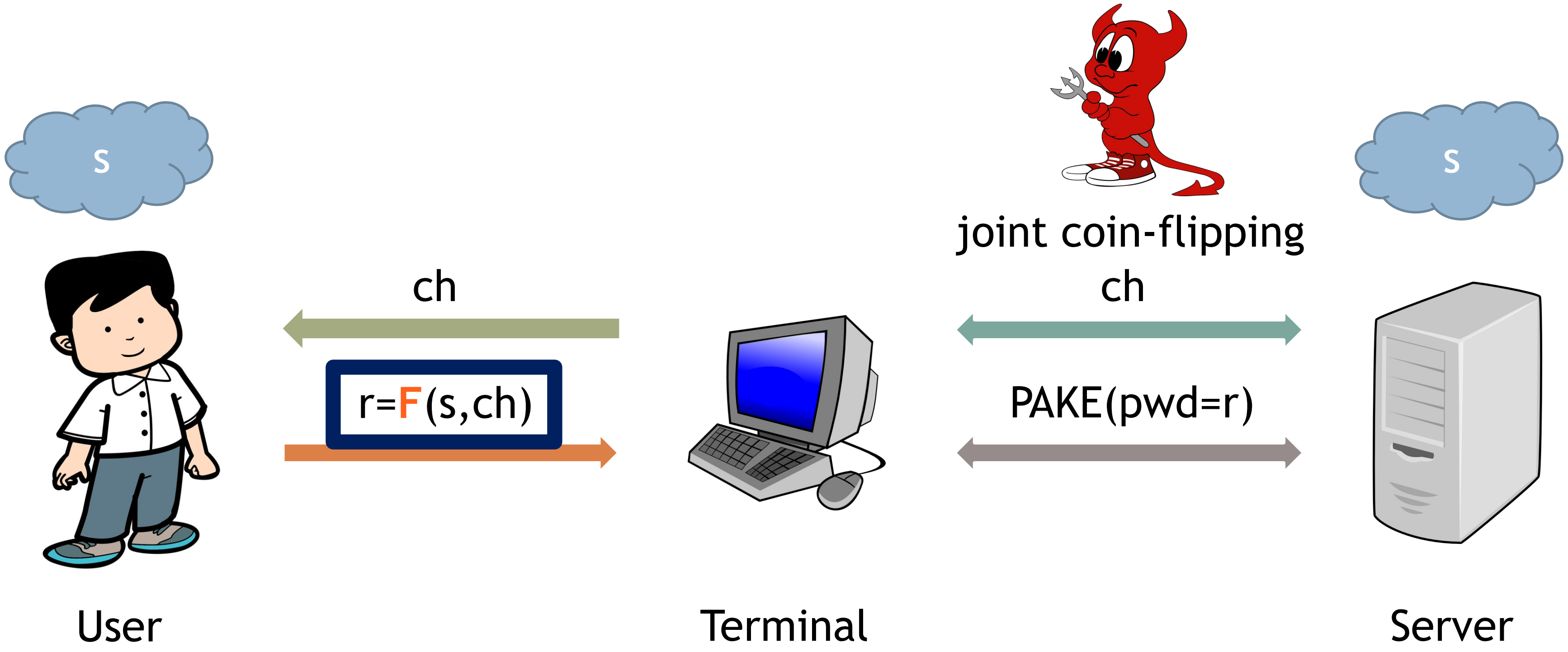
- Is this protocol secure? No!
  - **Replay** any challenge observed before to run a **fake server**.

Refine Basic Idea



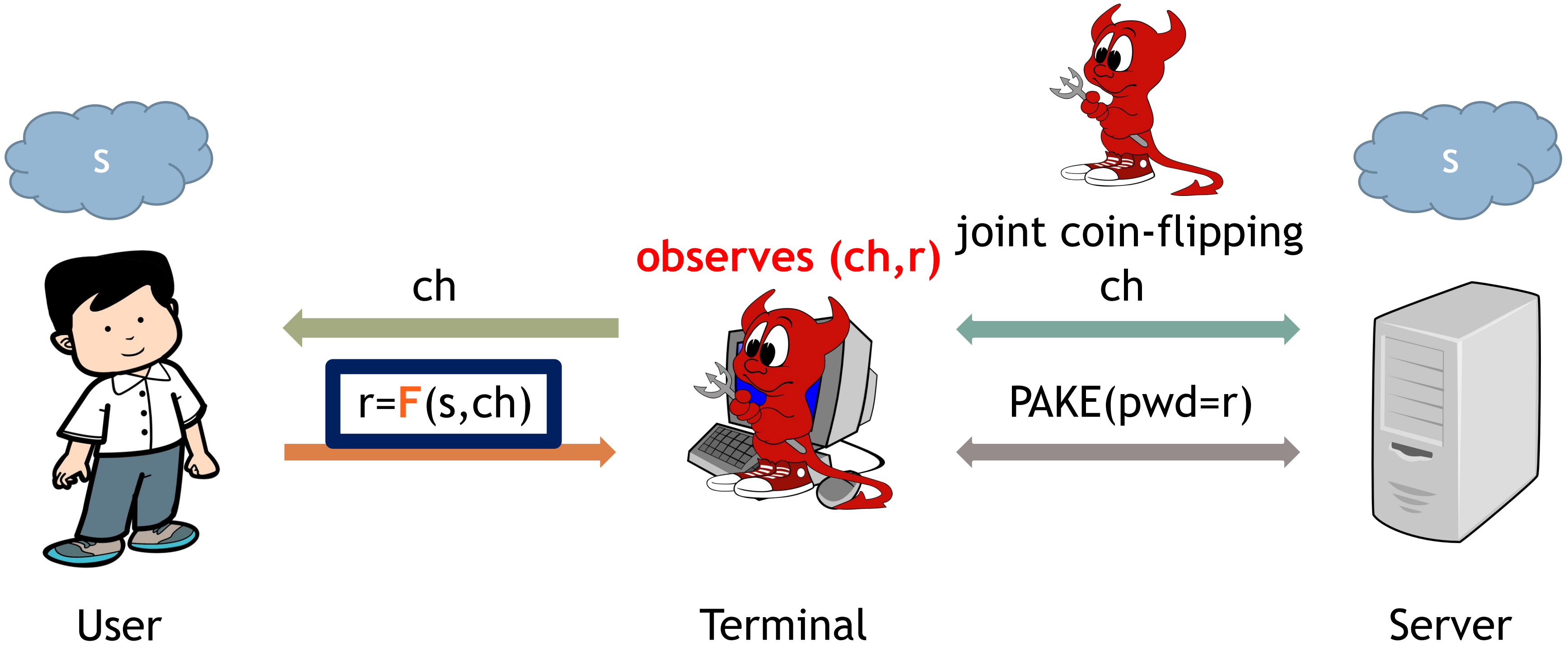
- How to prevent replay attack? **Joint coin-flipping**!
  - Uses commitment scheme to guarantee the random challenge is determined by **both** the terminal and the server.

Human-Compatible (HC) Function



- How to construct the **Human-Compatible (HC)** function F?
  - human-readable & writable & computable…

HC Function Security Model



- **Unforgeability**: Given (ch,r) pairs (some of which may be **adaptive**), the attacker can not **forge** the response to a **new random** challenge.

HC Function Instantiation

- **Only-Human** HC function
  - 😀 The user requires nothing but his/her **brain**.
  - 😱 Hard to construct:
    - too simple: easy to break
    - too complex: hard for human users to compute
- **Token-Based** HC function
  - 😱 The user requires an **additional device** such as RSA SecurID.
  - 😀 Very easy to get:
    - E.g., pseudorandom function (PRF)

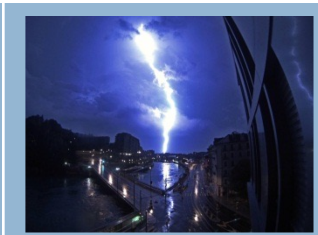
Only-Human HC Function Instantiation

- Human-computable function proposed by [BBDV16].
- In their construction (recall r=F(s,ch)):
  - challenge ch = several sets of numbers (represented by images)
  - response r = several digits
  - long-term secret s = random mapping from images to digits
- To use their function, need to show:
  - s is **human-memorizable**, F is **human-computable**.
  - HC function in [BBDV16] is **secure** in our model.

- The following 8 slides are adapted from the presentation by [BBDV16].

adapted from [BBDV16]'s presentation

Long-Term Secret / Random Mapping

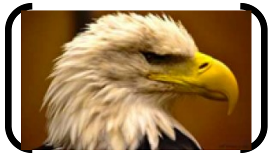
Image I			…	
σ(I)	9	3	…	6

- Random Mapping
  - $\boldsymbol{\sigma}$: $\{I_1, \dots, I_n\} \rightarrow \{0,1,\dots,9\}$
- Hard to memorize
  - mnemonics to help the user

adapted from [BBDV16]'s presentation

Long-Term Secret / Random Mapping

Mappings:

σ() = 2

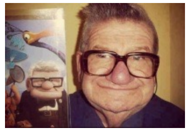
σ() = 6

Mnemonics:

adapted from [BBDV16]’s presentation

Challenge / Sets of Images

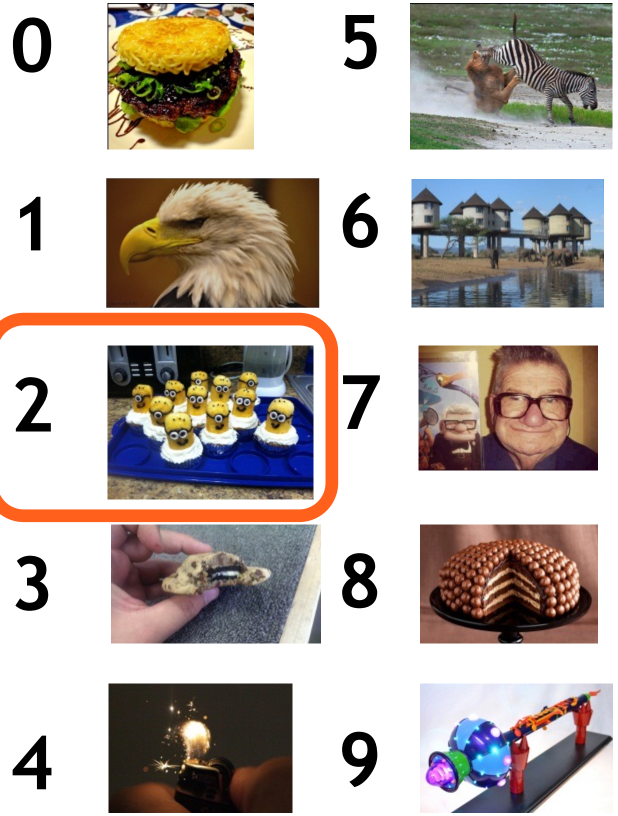



0 
5 
1 
6 
2 
7 
3 
8 
4 
9 

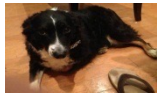
adapted from [BBDV16]'s presentation

Computing the Response r=F(s,ch)



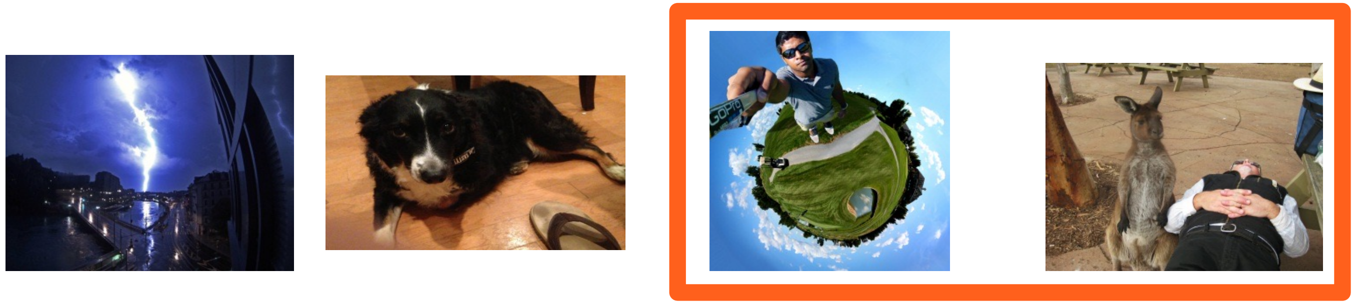
Computing the index:

σ () + σ () mod 10

= 9+3 mod 10 = 2

adapted from [BBDV16]'s presentation

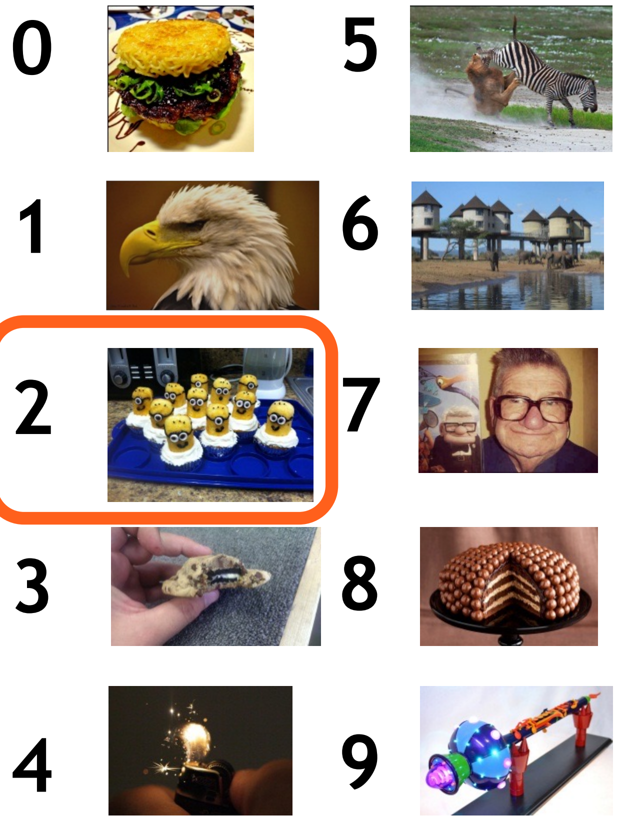
Computing the Response r=F(s,ch)



Compute the final digit:

non-linear against GE

= 7 + 4 + 5 mod 10 = 6



adapted from [BBDV16]'s presentation

Computing the Response r=F(s,ch)




Username: shan

Password: 6

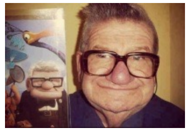
0 

5 

1 

6 

2 

7 

3 

8 

4 

9 

adapted from [BBDV16]’s presentation

Computing the Response r=F(s,ch)




Username: shan

Password: *5

0 

1 

2 

3 

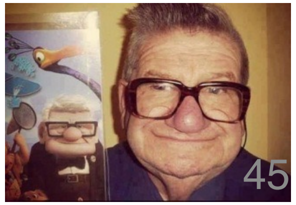
4 

5 

6 

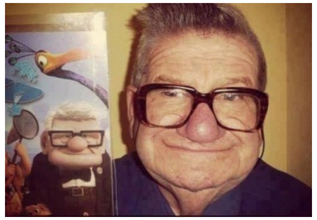
7 

8 

9 

adapted from [BBDV16]'s presentation

Computing the Response r=F(s,ch)




Username: shan

Password: **9

0 

1 

2 

3 

4 

5 

6 

7 

8 

9 

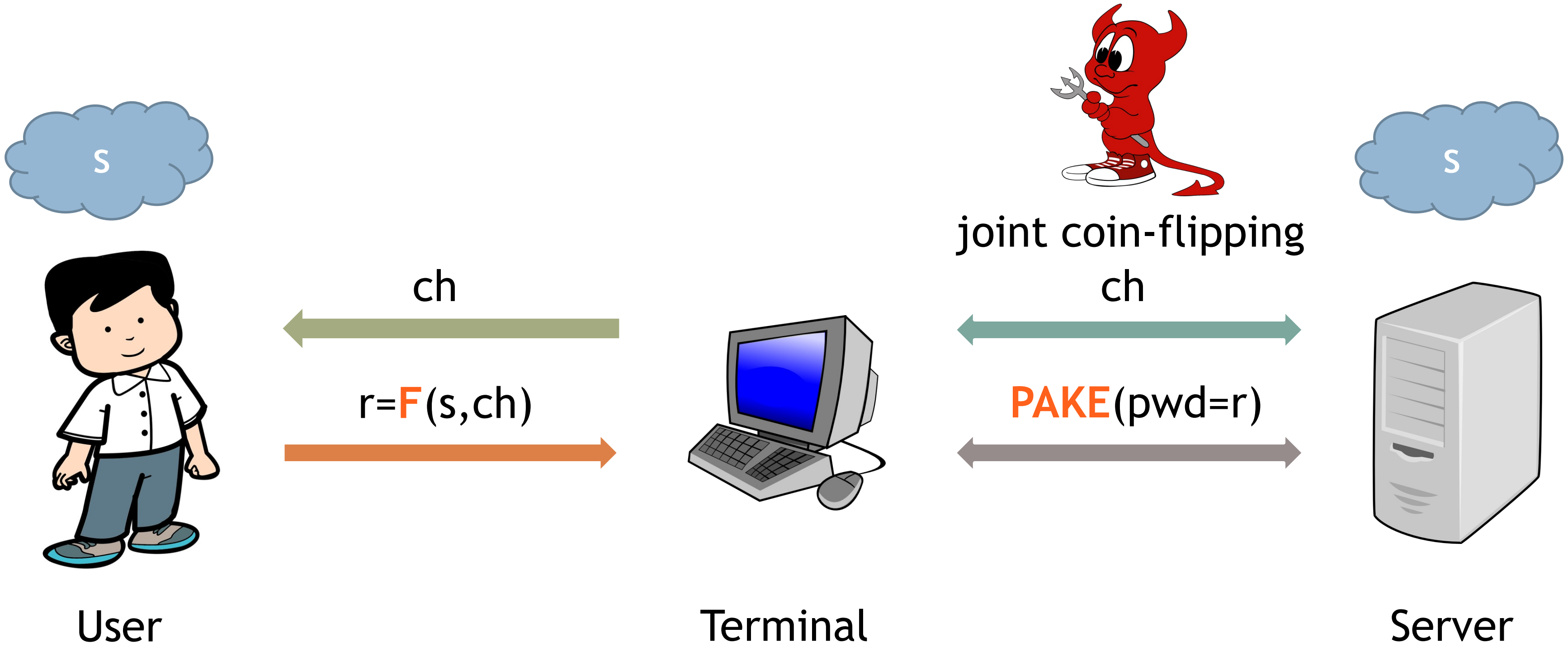
Usability

- Main Issue: Is the secret mapping human-memorizable?
  - Entropy is huge (but expected): 10^n possible mappings.
  - Usability experiment: n=100 images in 2 hours.
- The function in [BBDV16] is not perfectly suitable for humans. But functions with better usability may be proposed in the future.
- The **main contribution** of our HAKE protocol is to **provide a framework** that can allow for **any** secure HC functions.

Only-Human HC Function Security

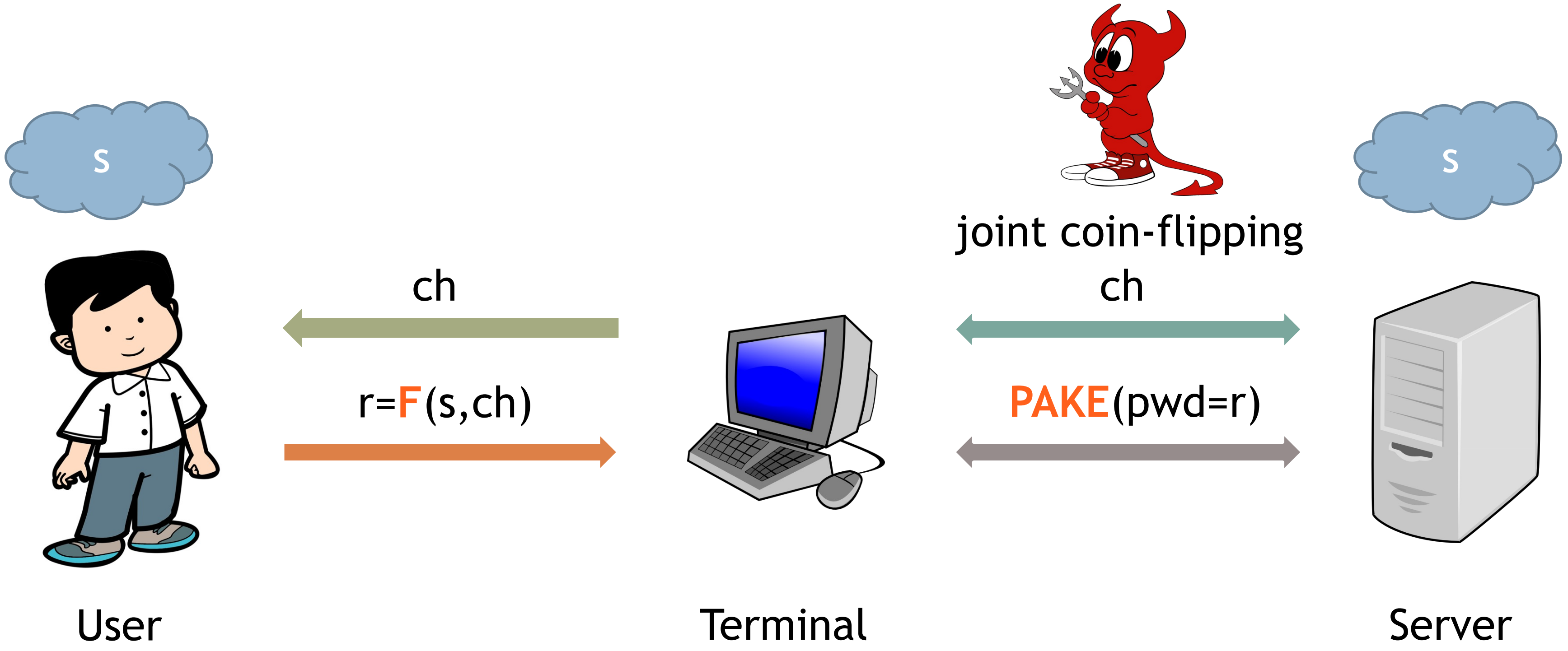
- [BBDV16] HC function security:
  - **Unforgeable** given not too many **random** challenge-response pairs
  - Based on the hardness of the random planted constraint satisfiability problems (RP-CSP)
- In our setting:
  - Thanks to PAKE, random challenge-response pairs are only observed from **compromised** sessions instead of all sessions.
  - We proved an extended security theorem to tolerate a **limited** number of **adaptive** challenge-response pairs.
  - HC function security is also based on an assumption similar to the **one-more unforgeability** assumption [BNPS03].

Basic HAKE



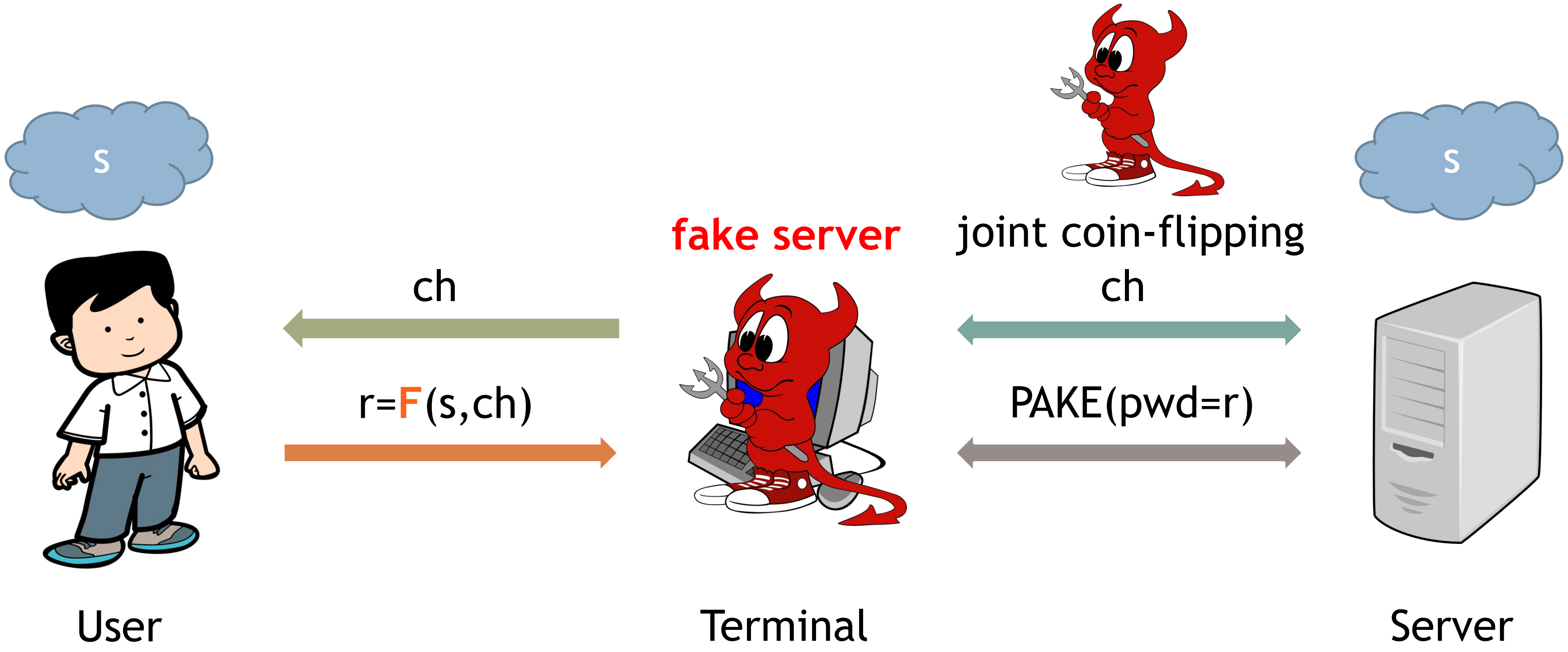
- First generic HAKE protocol!
  - secure Only-Human HC function F & secure PAKE

Basic HAKE



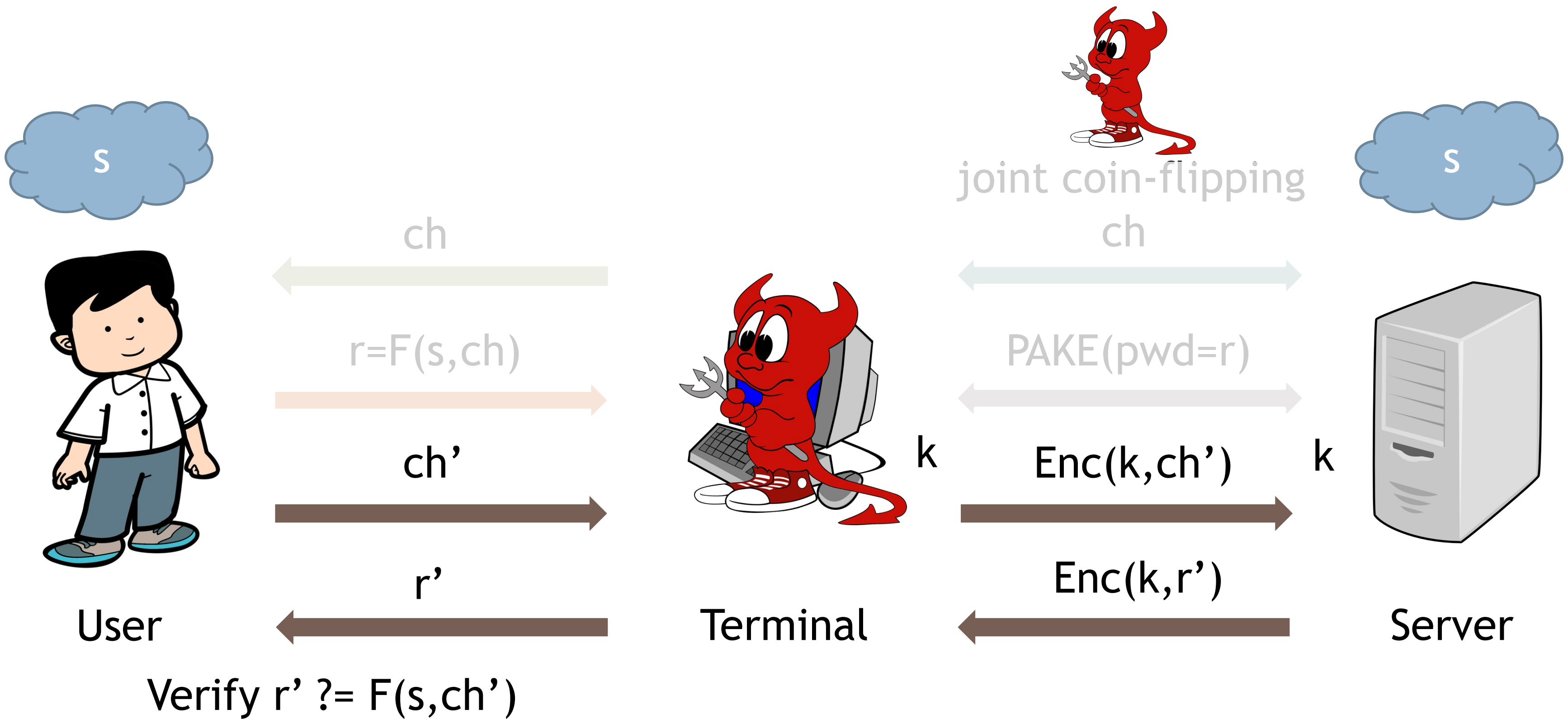
- Looks great! Are we done?

Basic HAKE



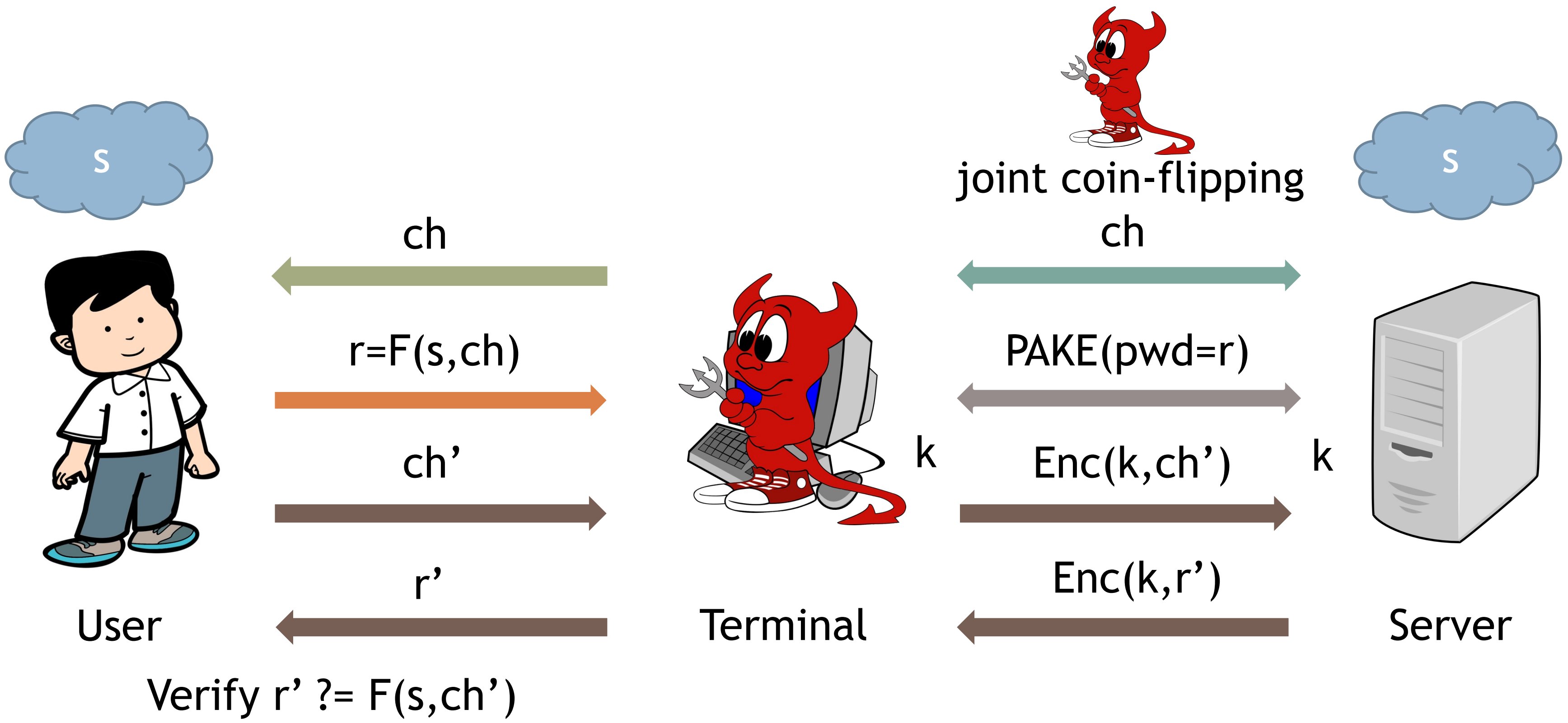
- If the terminal is fully controlled by the attacker:
  - many **adaptive** (ch,r) pairs may reveal the long-term secret s
  - need **explicit authentication**

Basic HAKE + Additional Round



- Introduce an **additional confirmation round** to Basic HAKE.

Confirmed HAKE

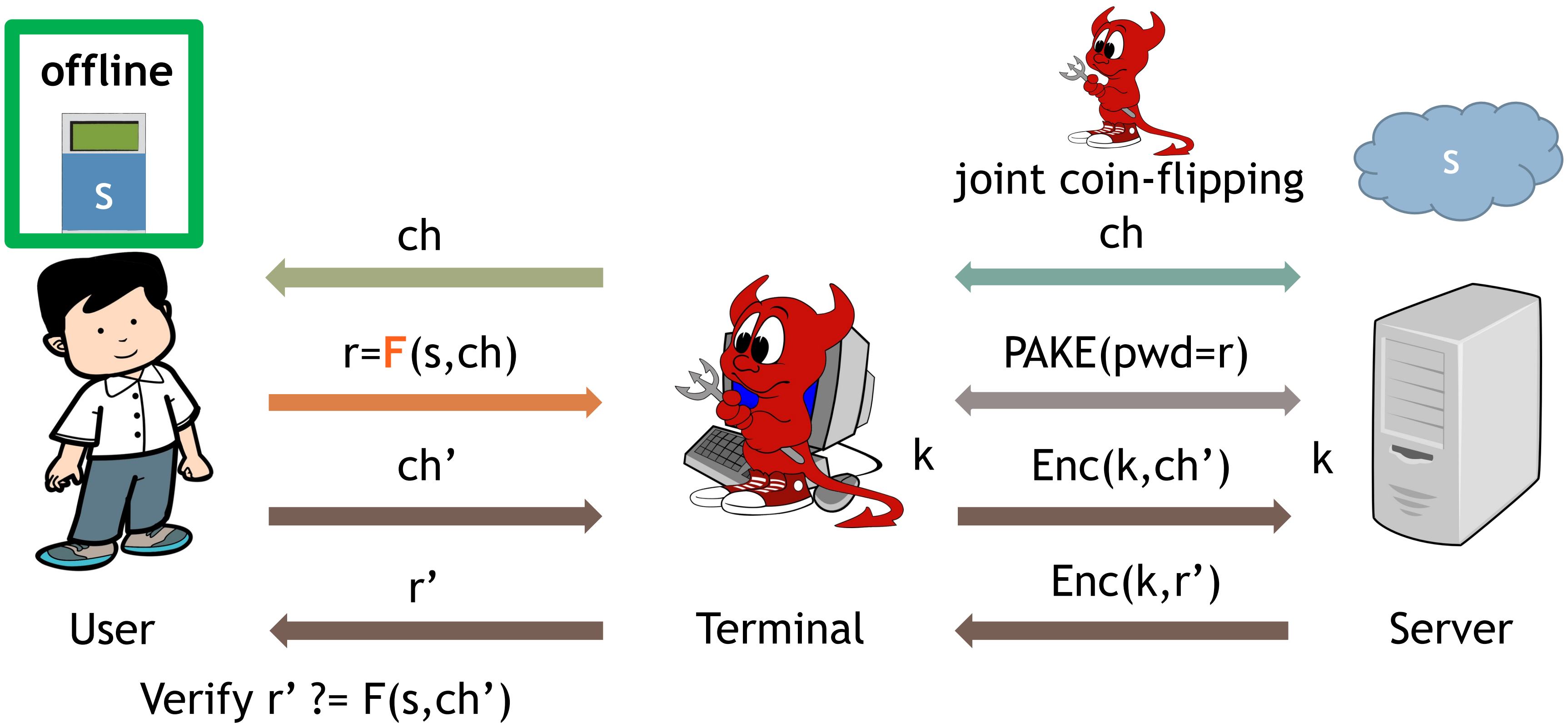


- **Detect** the compromised terminal & **Authenticate server**

Confirmed HAKE

- **Theorem**. Confirmed HAKE is **secure** if
  - Only-Human HC function F is **unforgeable** ([BBDV16])
  - PAKE is **secure** (EKE [BM92] in ideal-cipher model)
  - Authenticated encryption is **secure** (Encrypt-then-MAC)
  - Commitment scheme is **secure** (H(m,r) in the RO model)

Confirmed HAKE

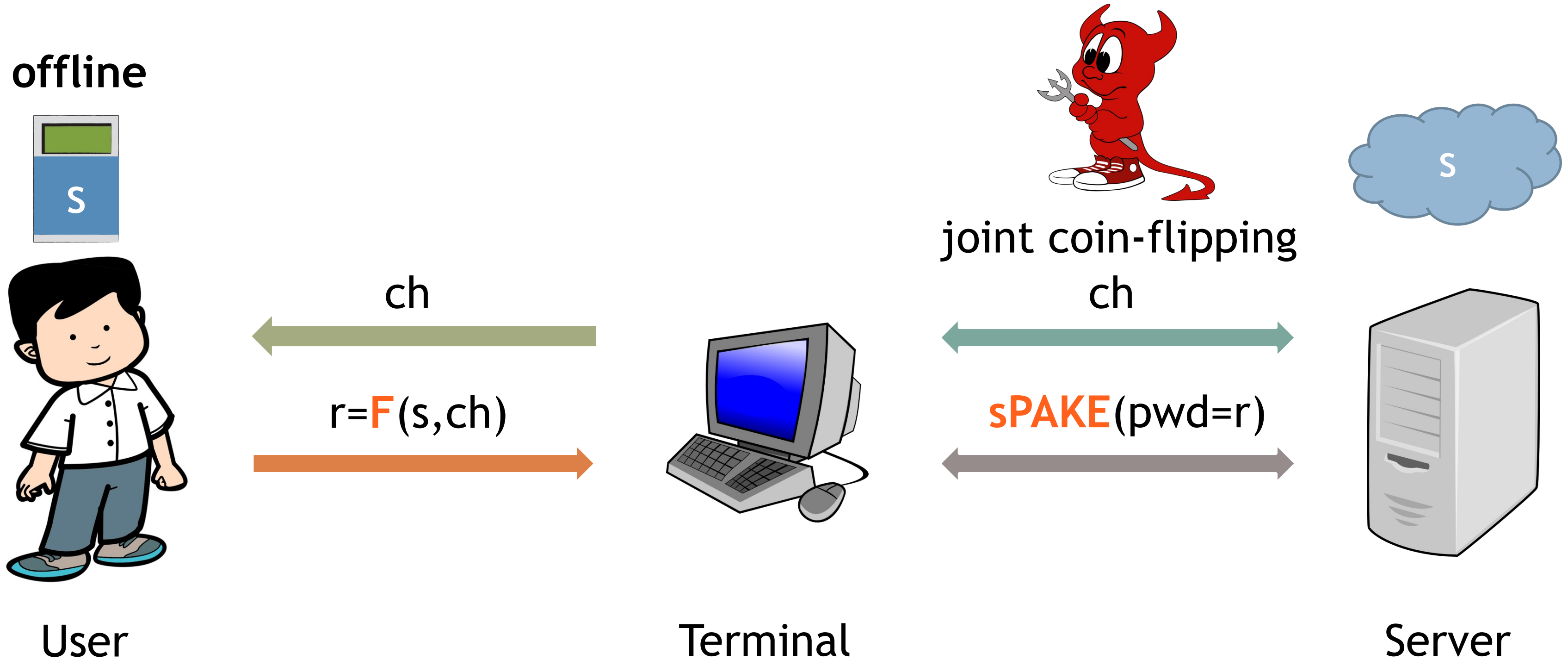


- What if the user has an **additional device**?

Device-Assisted HAKE

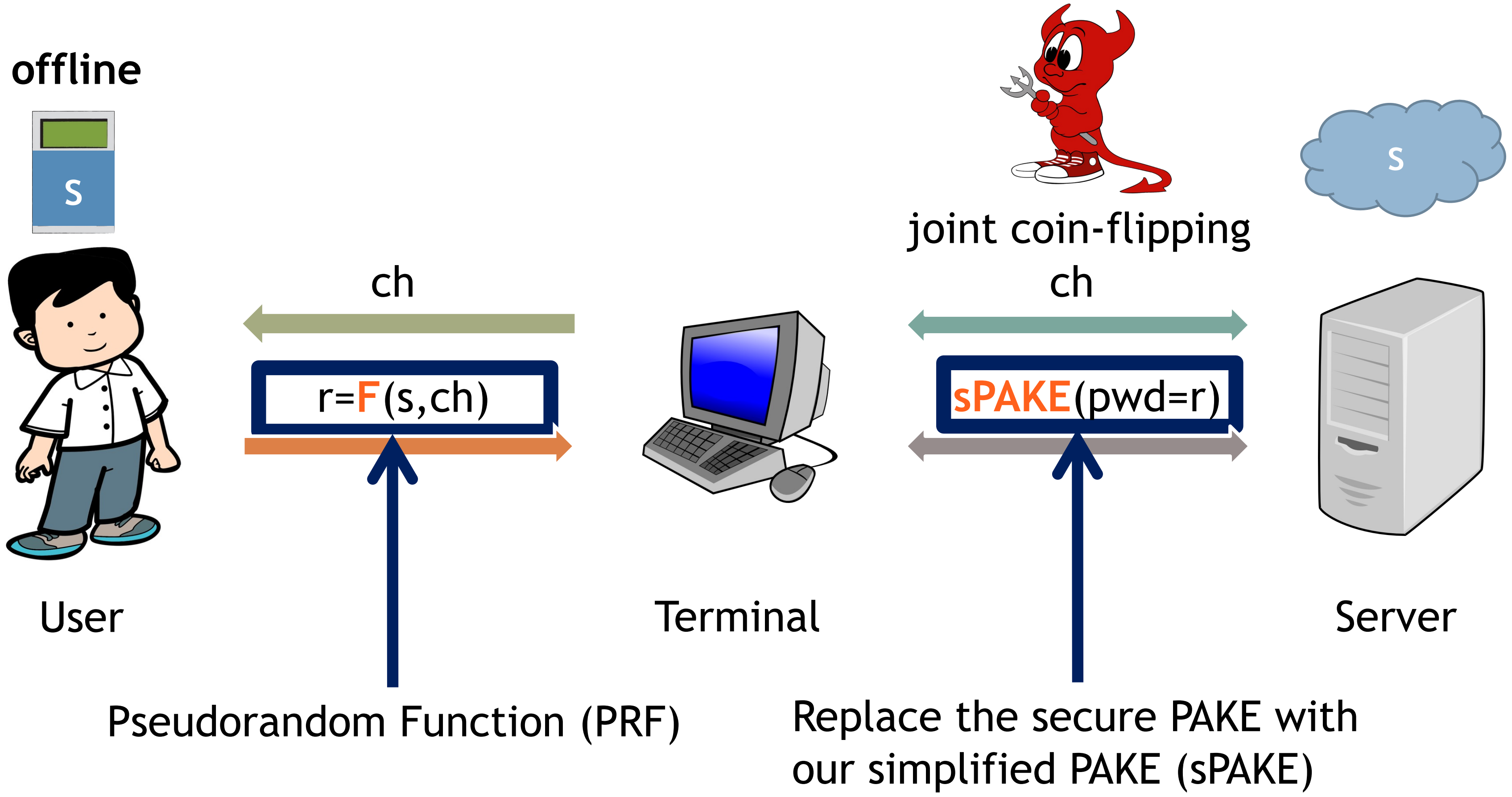
- With an additional device, we can instantiate F with Token-Based HC functions, e.g., pseudorandom function (PRF).
- Strong security: PRF is unforgeable given **computationally unlimited** number of **adaptive** challenge-response pairs!
  - no need to hide the responses (allows for a simplified PAKE protocol with weaker security)
  - no urgent need for explicit authentication (less rounds)

Simplified Basic HAKE

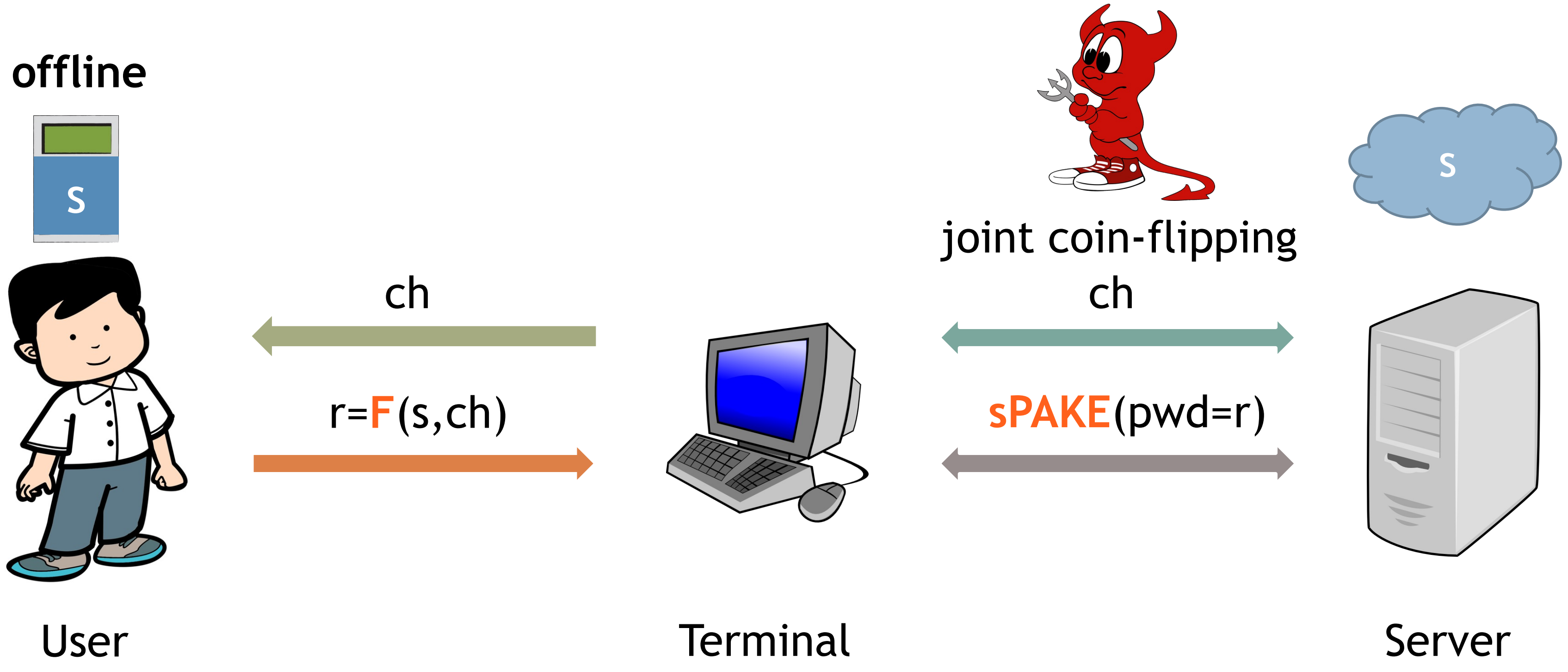


- Initial Device-Assisted HAKE protocol.

Simplified Basic HAKE

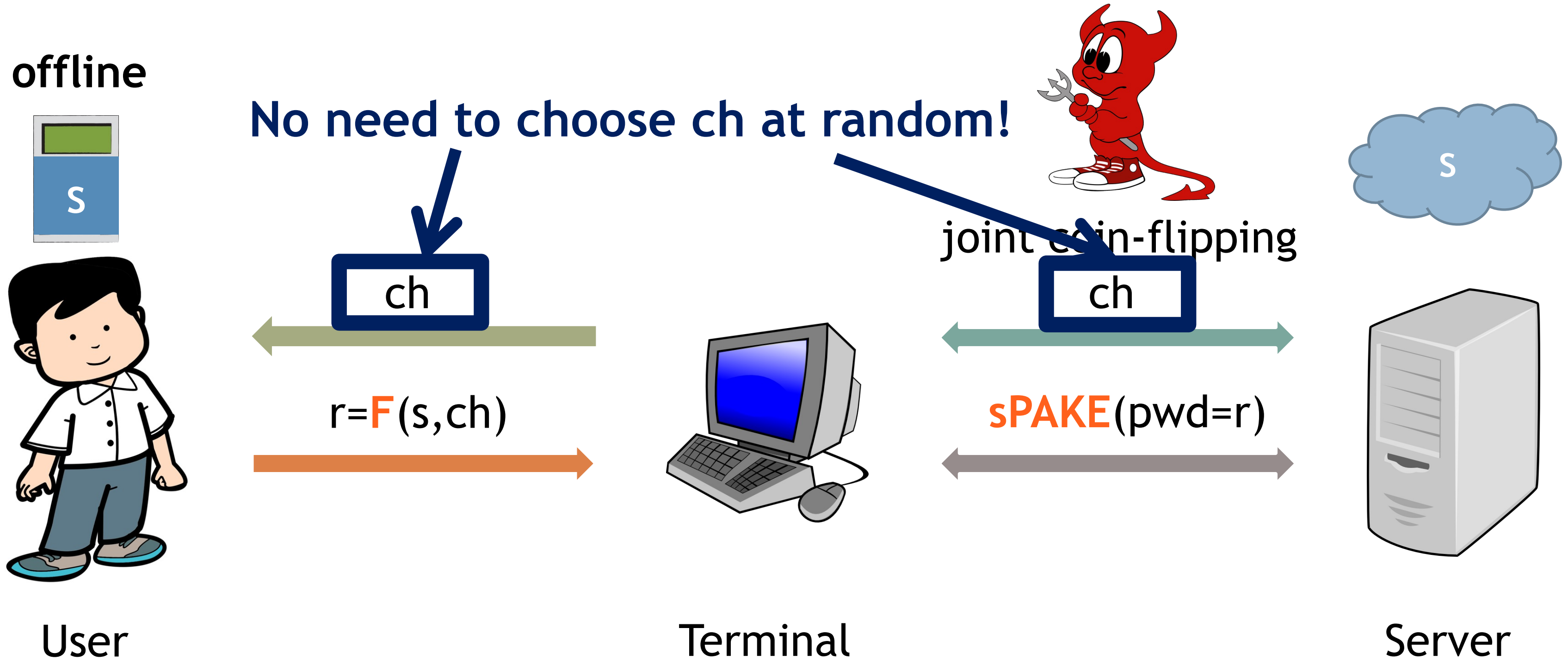


Simplified Basic HAKE



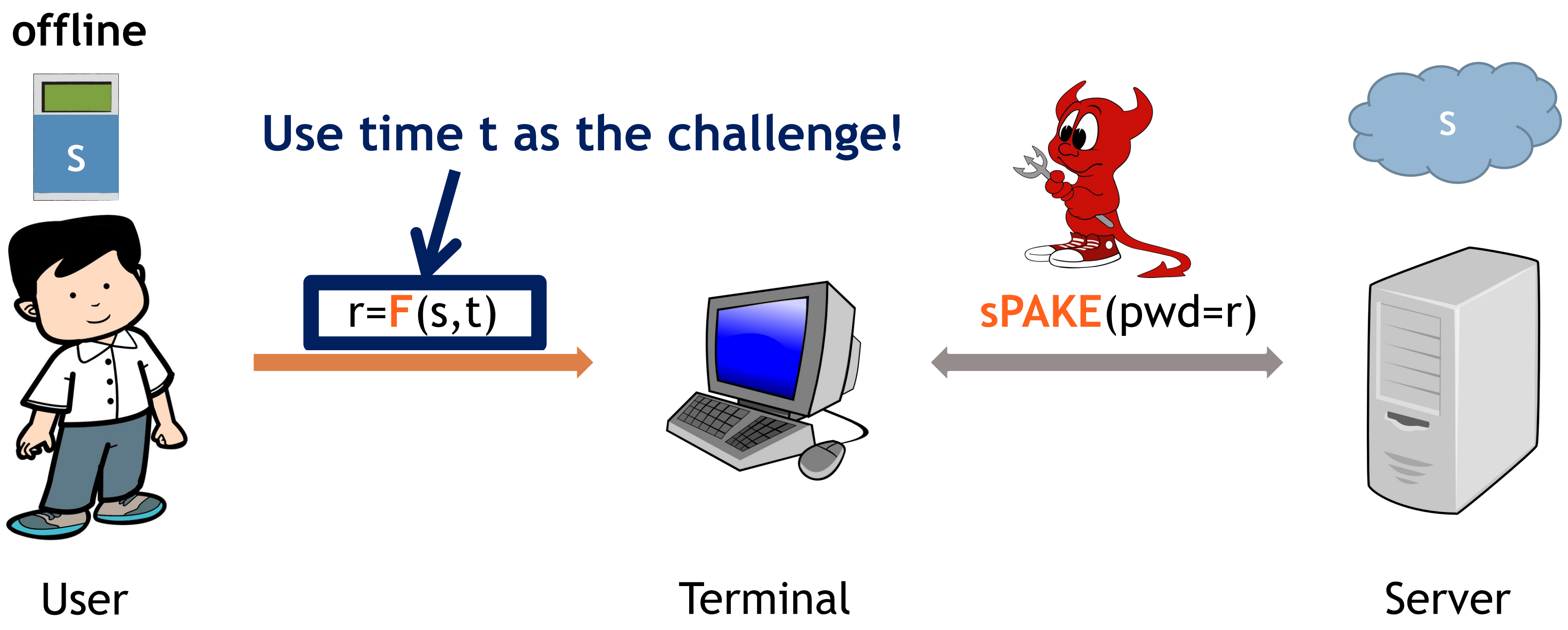
- Can we simplify this protocol further?

Simplified Basic HAKE



- Can we simplify this protocol further? Yes, we can!

Time-Based HAKE



- Very simple protocol!
  - Delay depends on the length of a single timeframe (usually several seconds).
  - Computational load is **30%** less than the most efficient one-time-PAKE [PS10, AP05].

Summary

- We proposed the **first** user authentication and key exchange protocols that can tolerate **strong corruptions** on the client-side.
  - Basic HAKE, Confirmed HAKE.
- We proposed **very efficient** Device-Assisted HAKE protocols that are also secure in case of **strong corruptions**.
  - Simplified Basic HAKE, Time-Based HAKE.

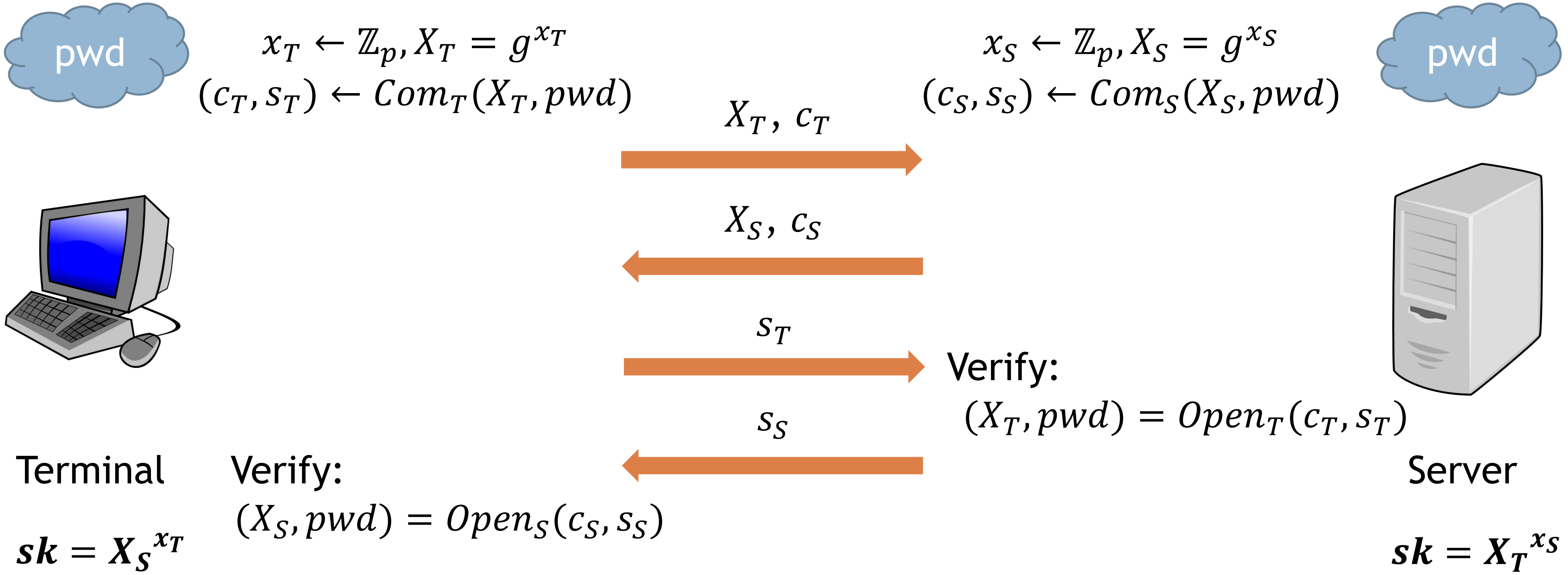
Open Problems

- Find Only-Human HC functions that can tolerate more adaptive (ch,r) pairs.
- Prove the security of the HC function in [BBDV14] without the one-more unforgeability assumption and improve its usability.
- Design a coin-flipping protocol directly between a human user and a server (to prevent adaptive (ch,r) pairs).
- Build an asymmetric version of the HAKE protocols (similar to the verifier-based PAKE) where no long-term secret is stored on the server.

Any Questions?

- Thanks!

Simplified PAKE (sPAKE)



- Diffie-Hellman + commitment scheme (more efficient, no encryption)

Time-Based HAKE vs One-Time-PAKE

Scheme	Flows	Terminal		Server		Communication Complexity
		exp	H	exp	H	
1(SPAKE1)	4	3	1	3	1	4λ
Time-Based HAKE	4	2	2	2	2	10λ

- The computational load is reduced by ~30% from the most efficient one-time-PAKE [PS10, AP05].
- Relaxing the PAKE security properties allows a significant efficiency gain.