Integrating mutual human being – machine authentication into TLS (PAKE)

Might we best prepare a modularized interface for flexible authentication mechanisms?
Concept ideas/suggestions for discussion in the CFRG/TLS working groups.

Secure mutual authentication for remote human-machine interfaces (HMI)

Outline of this presentation:

1.) Problem space has at least three “dimensions” that need to be considered:

- HMI User expectation for secure logins
- Software architecture / Maintainability requirements
- Security architecture / Security proofs / Security assessment

2.) How a modularized approach using UC-secure subcomponents / subprotocols such as CPace and AuCPace might be able to provide a manageable migration path to PAKE and flexible authentication of human individuals also beyond PAKE.

HMI User expectation – 1 -

- Today, most important remote HMI tool: Web server
- Presently, most important authentication method: Logins based on username/password
- In the future other authentication mechanisms might become more important / interesting:
  - We might want to combine username/password with authentication hardware (“company badge”)?
  - What about fingerprint/QR-Code based authentication for web server logins in consumer applications?
  - Might it be nice to use existing (e.g. RADIUS) authentication services for TLS session authentication?
- Common feature: One or more components of the authentication might be of a low-entropy type.

- Today: Often solutions for two-factor authentication systems require complicated HMI handling (e.g. entering PIN numbers from a hardware token). Not seamlessly integrated in browsers/TLS.
- Neat integration into web servers and flexible choice of the authentication mechanism by the server device might become highly desirable in the future.

HMI User expectation – 2 -

- Today the user is expecting a login sequence
  - Establish connection to remote web server
  - Enter authentication credentials upon request
  - Obtain access
  - Possibly re-authenticate for starting critical operations
    ("Do you really want to erase all data? Please re-enter password.")

- Security-wise, this user expectation has its justification. The normal operation should be that user credentials are entered only upon explicit request, i.e. not in advance as preparation of a possible operation in the future. (=> Consequences for a TLS handshake)

- After successful login, users also need to be able to manage the accounts. (Change passwords, add users, manage permissions, etc.)

HMI User expectation – 3 -

- More and more end users will have to set up “web-server”-style remote logins for the remote HMI interface of their IoT devices.
- Many such applications will mandatorily require good security.

- Even experts sometimes struggle with integration of servers in today’s Web-PKI

- We need both, a secure and convenient solution that should not solely rely on a well-managed Web-PKI for such “end-customer-owned” server devices.
- Web-PKI based security might not serve many IoT use cases.

Software structure – 1 – (TLS side)

- TLS today should be considered a mechanism for securing machine-to-machine interfaces.

- Assessment B. Haase:
  - Today’s TLS environments might not be prepared to handle the complexity that comes with user account management, add/remove users, invoking HMI user dialogues, etc.
  - We would be able (with some pain) to integrate the essential username/password – interfaces in TLS. But when we start with future more secure / more convenient authentication mechanisms (Fingerprint? 2F Password+Smart-Card Badge) as basis for TLS authentication, the complexity might explode.

- Suggestion B. Haase:
  If we want to allow for a flexible human-user authentication with TLS, we might want to prepare some kind of modularized system?

Software structure – 2 - (server system side)

- Security-wise password handling should be kept away the normal "application" code.
- For managing accounts on devices, many systems already have special authentication submodules written by people with some security background. (E.g. PAM on Linux/Sun).
- On the server-side, a remote TLS-protected login process should best refer password handling to a "PAM-style" submodule.
- A TLS/PAM-based user authentication could be helpful for a wide range of applications: Remote shell / Version management tools such as GIT / Web Servers

- For TLS integration strategy, we should consider the needs of the "PAM-style" system partner
  - Password verifiers should also be suitable for use with local (i.e. not remote) logins
  - Password verifiers should not have excessive size
  - Different levels of granularity for attributing user authorizations should be possible

Software structure – 2 - (client system side)

- On the client side, we need platform-specific GUI controls, e.g. for entering passwords and user names.
- GUI systems will be highly platform / OS-specific

- The TLS implementer probably does not want to deal with this aspect.

- Handling of the GUI masks for entering user names and accounts should best not be under control of the “application” but handled by a special security software component.

Security dimension

- In the future, security systems, such as authentication of human individuals will become more and more complex.
- The attacker will always be targeting the weakest spot.
- Analysis / security proofs are complex, even for comparably “simple” systems, such as today’s TLS which focuses on certificate/PSK authentication.
- Analysis / security proofs will become even more difficult for more complex composed authentication systems.

- We might need special strategies and modularization for the security analysis. We might want “Security LEGO bricks” for human operator authentication.
- Pre-analyzed secure components which don’t loose their security guarantees when being arbitrarily composed in larger systems ?
  Universally composable protocols!

2.) How a modularized approach might provide a migration path

- Special properties of AuCPace und CPace
- How a modularized user-authentication eco-system for TLS might become manageable.

Special properties of the AuCPace / CPace construction

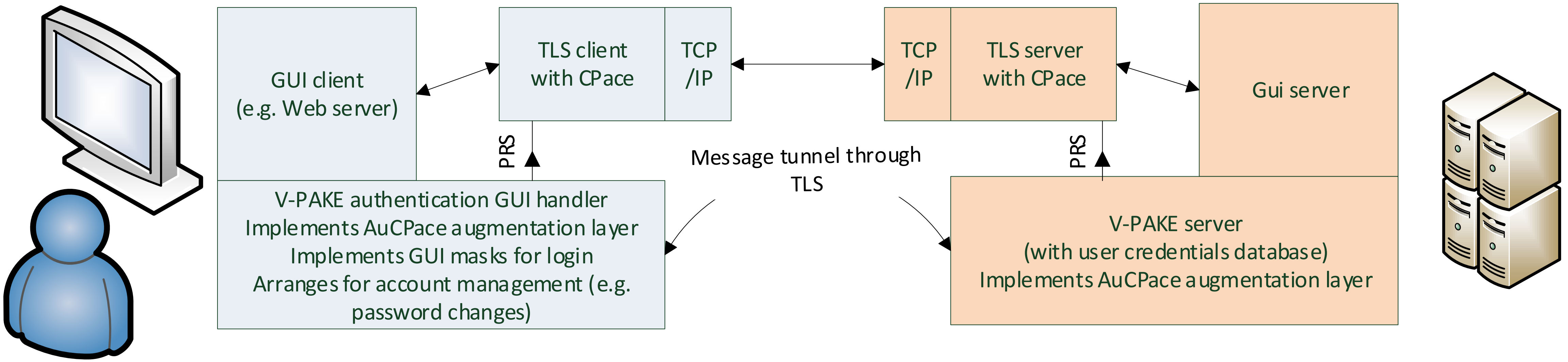
- Unlike other proposals to CFRG PAKE selection, AuCPace / CPace is in itself a modular construction.

1. AuCPace augmentation layer calculates a session-specific ephemeral string "PRS" which involves the low-entropy password and salted hashing
2. AuCPace then invokes CPace with "PRS" as parameter
3. CPace comes with an independent UC security proof.
   CPace arranges for session keys, forward secrecy and implicit authentication of "PRS" and fends of relay attacks.
4. Subsequently explicit key confirmation may optionally be carried out.

Suggestion for augmented PAKE (V-PAKE)

Client

Server

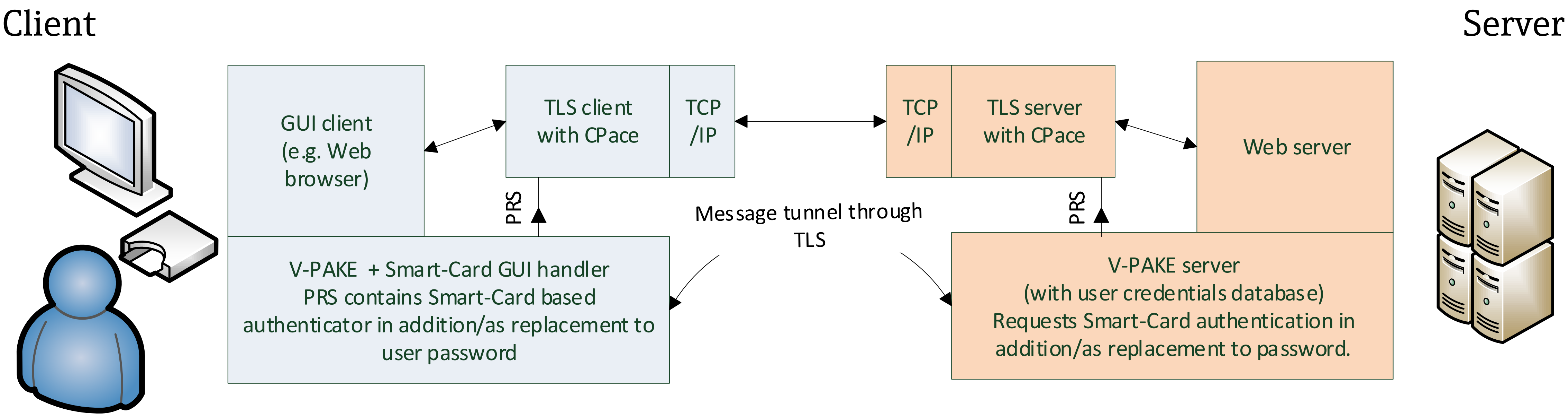


TLS implements a tunneling mechanism for authentication message exchange

TLS implements UC-secure balanced PAKE CPace

UC-Secure "augmentation layer" establishes ephemeral PRS on both sides using tunneled information messages in the TLS handshake and post-handshake phases.

Suggestion

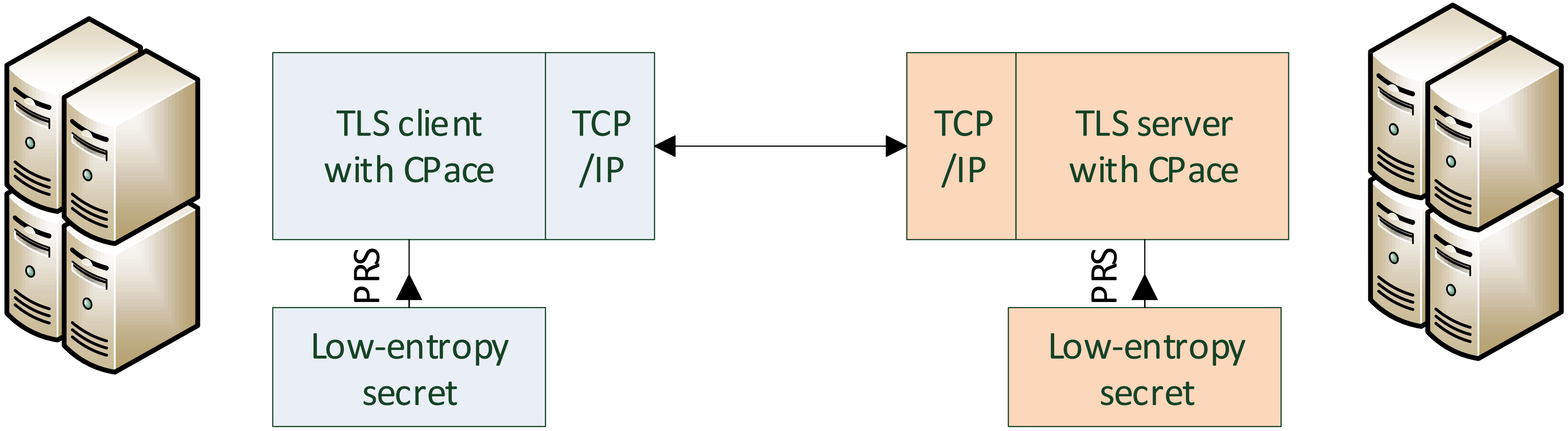


Future extensions (e.g. “UC-Secure smart-card-based authentication”, “UC-Secure fingerprint-based” authentication, RADIUS-server based authentication) could use the same TLS-CPace APIs for future extensions without need of modification of the TLS stack core.

Different ways of calculating the PRS input to CPace will be possible.

TLS-CPace just manages session confidentiality, integrity, forward secrecy and authenticates PRS.

Machine-Machine Use-Case



- Machine/Machine interfaces could use CPace without an augmentation layer based on a pre-shared secret “PRS” which may be of low entropy.

Summary

- Too neatly integrating user interfaces into TLS might generate trouble.
- Main new features desired for TLS for mutual authentication of human users with computer devices might be a “user authentication message tunneling” mechanism and a balanced PAKE?
- If a secure authentication based on a low-entropy ephemeral secret PRS would be available in TLS, *many* use-cases could be implemented.
- This “low-entropy secret session authentication” in TLS should best come with universal composability guarantees in order to allow for manageable security proofs of larger systems.

- CPace + AuCPace (ia.cr/2018/286) with their security analysis in the UC framework might allow for such a flexible and extendable approach.

Thank you for your attention.

Please share your thoughts, criticism and suggestions with us.
We are looking forward to starting a discussion with you.

Add-on slides

Special consideration regarding the Web-Browser use-case
Wearing the hat of the TLS designer, lessons learned from PAKE integration in the Smart Blue App.

Backup slides

1.) Considering the Web-server Use-Case

TCP-Port numbers to use for browsers with TLS-PAKE (username/password)

Two options:

1. Use port 443 (https:/) also for http over TLS-PAKE
   - The browser cannot know in advance, that a user name will be required for TLS handshake
   - The GUI dialogue for "username/password" would pop up only after a hello-retry from the server.
2. Allocate a new port number (e.g. httpp:/) for TLS-PAKE with username-password
   - When seeing httpp:/ in the address field of the browser, the browser could show the GUI dialogue for "username/password" prior to the TLS handshake
   - The "user name" information could be given already in the TLS "client hello"

TCP-Port numbers to use for browsers with TLS-PAKE (username/password)

Two options:

1. Use port 443 (https:/) also for http over TLS-PAKE
   - The browser cannot know in advance, that a user name will be required for TLS handshake
   - The GUI dialogue for “username/password” could pop up only after a hello-retry from the server.
2. Allocate a new port number (e.g. httpp:/) for TLS-PAKE for username-password
   - When seeing httpp:/ in the address field of the browser, the browser could show the GUI dialogue for “username/password” prior to the TLS handshake
   - The “user name” information could be given already in the TLS “client hello”

Approach 2.) might be manageable, if we only consider “PAKE with username/password”.
If we also want to consider “PAKE with username/password + Hardware token” we would need a third TLS port number httppht:/ ? … maybe a fourth port for a fingerprint-assisted TLS?

TCP-Port numbers to use for browsers with TLS-PAKE (username/password)

Two options:

1. Use port 443 (https:/) also for http over TLS-PAKE
   - The browser cannot know in advance, that a user name will be required for TLS handshake
   - The GUI dialogue for "username/password" could pop up only after a hello-retry from the server.
2. ~~Allocate a new port number (? httpp:/ ?) for TLS-PAKE for username-password~~
   - ~~When seeing httpp:/ in the address field of the browser, the browser could show the GUI dialogue for "username/password" prior to the TLS handshake~~
   - ~~The "user name" information could be given already in the TLS "client hello"~~

Assessment B. Haase:
Future-proof and extendable human-user authentication could only be based on option 1.)

We should assume that we could not provide the user name in the first TLS "client hello".

Endress+Hauser

Consequences if the first client hello could not include the user name

- Further communication rounds are necessary for PAKE with web browsers ☹.
- Is this a problem in practice? No!
  - Q: What is the motivation for few round-trips for TLS on human user interfaces?
  - R: User experience, responsive GUI interfaces!
- In the PAKE use-case, the GUI ***will*** be responsive!
  The “username/password” dialogue will pop up immediately after reception of a “hello retry” message. This is exactly what the user expects!
- In practice, the main delay factor might be the human user because of the time that he takes for typing the password.
- Regarding usability / GUI requirements, the number of round-trips should be considered much less critical than for the typical WEB-PKI setting for web browsers.

Consequences if the first client hello could not include the user name

- Further communication rounds are necessary for PAKE with web browsers ☹.
- Is this a problem in practice? No!
  - Q: What is the motivation for few round-trips for TLS on human user interfaces?
  - R: User experience, responsive GUI interfaces!
- In the PAKE use-case, the GUI ***will*** be responsive!
  The “username/password” dialogue will pop up immediately after reception of a “hello retry” message. This is exactly what the user expects!
- In practice, the main delay factor might be the human user because of the time that he takes for typing the password.
- Number of round-trips should be considered much less critical than for the typical WEB-PKI setting for web browsers.

(Note that for the low-entropy PSK use case for machine-machine interfaces we would be keeping the 1-RTT feature of TLS 1.3! PRS is known at “client-hello” time.)

Backup slides *added after the IETF 105 CFRG session*

2.)

"Lessons learned from integrating augmented PAKE into the E+H SmartBlue App"

...or...

"Why we might best be keeping TLS 1.3 state machine as simple as possible."

PAKE integration in to the E+H SmartBlue App / Lessions learned

- In 2017 Endress + Hauser has rolled out a remote HMI App “SmartBlue” whose main security feature is a PAKE-based certificate-less security layer “AEM” (Authentication and Encryption Module).
- AEM already integrates what is detailed in the AuCPace paper (ia.cr/2018/286)
- Similar to TLS, AEM has a „Record“ layer, Symmetric cipher suite activation, etc. and we also have some equivalent to „PAM“ in the field device and asynchronous API access to flash memory for security logbooks, etc. .
- One main observed challenge for integrating PAKE: Complexity of the state-machines
- Main lesson learned:
  It is possible to manage the complexity, but we needed to split the whole system into manageably small sub-state machines.

PAKE integration in to the E+H SmartBlue App / Lessions learned

- The handling of the user interface resulted in a large number of asynchronous event sources and states.
  - Asynchronous interface to the GUI frontend
  - Read after Write for the security Logbooks.
  - Session timeout handling
  - Rate-Limiting
  - „Too many failed logins, please retry in 15 minutes“ messages
  - „Prior to your successful login there were 3 failed login attempts“ message.
  - „Your password will soon expire, please change“ message.
  - „You logged in with a default password. We recommend a password change.“ message.
- State machine diagrams for the user interface and user credential handlings fills a wall paper.
- We needed a markup-language high-level definition for the state machines and code-generators for managing the complexity.

Comparison BlueConnect AEM / TLS

- In contrast to TLS, for AEM we had one team developing the code for both, clients and servers. We did not have backward compatibility nightmares.
- Still the complexity of the state machines was a challenge.

- For TLS the development-team situation is much more complex.
- We have much more implementation pitfalls, e.g. due to (broken) legacy implementations.

Our assessment:

From a the “TLS handshake sub-state machine” perspective, TLS 1.3 should probably aim at sticking to the 1-RTT model, also for PAKE.

Possible language problem for the term "Message flow"

For a TLS developers the term "Message flow" might combine two components

1. An additional data communication on the network channel
2. An additional complexity for the state machine and the handshake transcripts

In my opinion these might actually be two separate things.

- Adding some additional opaque communication on the network channel without any influence to the TLS state machine might be a smaller issue.
- Adding more complex state handling in the TLS 1.3. state machines should be considered a much more serious problem.

"1-RTT for TLS!" is not necessarily equivalent to "No additional network message!", if the communication is opaque to TLS handshake protocol!