Paper

BSBI – a Simple Protocol for Remote Verification of Identity

Adam Kozakiewicz and Tomasz Pałka

Research and Academic Computer Network (NASK), Warsaw, Poland

Abstract—The paper presents the design and the rationale behind a simple verification protocol for autonomous verification modules, and the architecture enabling use of such modules. The architecture assumes strict separation of all personal metadata and the actual verification data. The paper also describes a prototype implementation of the protocol and its extension enabling the state of the module to be monitored from the main system. The proposed design solves the problem of using advanced verification methods, especially biometric ones, in systems where direct implementation is not possible due to hardware incompatibilities, insufficient resources or other limitations.

Keywords—access control, authentication, biometric verification, network protocols.

1. Introduction

The security of any computer system depends to a large extent on the proper verification of the identity of its user. The access control policy used in the system is meaningless if the user is misidentified and allowed to work as someone else, especially if the user is not authorized to use the system at all. In most systems, this verification of identity is very simple, usually requiring the knowledge of a password. Naturally, systems with higher security requirements should employ more secure methods of verification. The password protection is actually quite good, there is no reason to eliminate it, but it should be accompanied by other methods, preferably employing hardware identification devices (tokens, cards, etc.) and/or biometric measurements.

In this paper we are dealing with the problem of applying advanced biometric verification methods in a system, which – for reasons stated in the next section – cannot implement them directly. We introduce a separate module, called BSB, responsible for performing the verification. We present the architecture of such a solution and the protocol used to connect the host system with the verification station.

Section 2 shows the background of this research, explaining the reasons why a separate verification module is needed in our solution. The section also presents a simple architecture which satisfies our requirements. As the architecture requires a special protocol for communication between the BSB and the host, we review the existing solutions in Section 3 and – having explained why none of them fit our needs – describe the protocol we designed, called BSBI, in Section 4. Section 5 describes our prototype implementation and Section 6 explains how it could be perfected in the future. We conclude with a short summary in Section 7.

2. Background, Assumptions and Design

The secure workstation for special applications [1] is a Linux-based system using visualization to process data from different security domains in separate environments. The guest systems can be either Linux- or Windows-based, but the main point of access control is the host system, based on Linux (specifically the project uses Red Hat Enterprise Linux). The task of verifying the identity of the user is performed at the host level. The built-in capabilities, including password-based verification are available. However, the system's high security level requires more advanced access control, preferably using several different methods in parallel.

Among the project goals is the demonstration of different authentication mechanisms [2], including hardware-based verification (e.g., token or card [3], [4]) and biometric verification [5], [6]. Specifically, we chose to implement iris recognition [7], [8]. This biometric modality is relatively reliable from the technical point of view, guarantees low false negatives and – more importantly – false positives.

The combination of iris recognition and Linux host operating system is unfortunately a significant compatibility problem. The commercially available specialized cameras for iris recognition rarely have drivers for Linux systems. Even the few that do, do not offer full functionality with those drivers. Furthermore, iris recognition requires quite advanced numerical analysis, available in the ACIrisSDK library developed at NASK [7], [8]. However, due to the driver issues described above, that library was written for the Windows system and porting it is not an easy task. Implementing iris recognition on the host system level would therefore be prohibitively difficult.

The difficulty is not the only reason against a host-level implementation. For security reasons, the host operating system should be minimal and based on well-tested software. The biometric processing is quite complex and works with externally provided data (photos of the iris). This makes it a potential weak spot in the system, especially since – as stated before – it would have to be new, implemented from scratch or ported in a non-trivial way. With no prior production use, even after rigorous tests this implementation would be likely to contain numerous bugs, some of which may be exploitable security vulnerabilities.

This situation left the designers with only two possible choices. The iris recognition would have to be performed on a Windows-based system, separate from the host. That system could be a virtualized guest or an external module. Since the workstation is basically designed as a minimal, secure virtualization environment, a hypervisor is already present on the host and the virtualization approach may seem sufficient. In fact, this is illusory. Implementation as a virtual machine would have several important consequences for the whole workstation, quite contradictory to the projects goals. The physical separation of virtual machines, postulated in the project, would require a whole physical CPU to be reserved for the biometric virtual machine[1]. A computer with two physical processors would then not be enough to run two protected, user-accessible virtual machines in parallel, as required. The alternative, deactivating the biometric VM when not necessary, would solve the issue, but greatly increase the verification delay.

The requirements of the biometric machine are also problematic. It would require full access to the camera, most likely connected via USB, and two-way communication with the host. The other virtual machines are not allowed to have any two-way communication with the host and their access to USB ports is heavily controlled (only special, cryptographically protected USB drives are allowed). Adding exceptions increases the probability of introducing vulnerabilities.

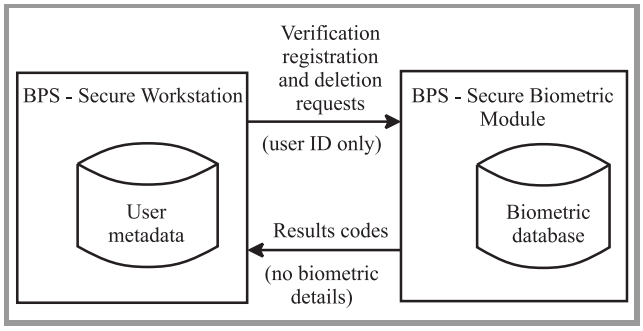

Fig. 1. The general architecture of the system. Note that the two types of data are kept separate and linked only with a numeric identifier.

For the above reasons, the final decision seems clear – the iris recognition will be performed on a separate physical machine. The system will consist of two separate modules, as shown in Fig. 1: the secure workstation and the verification station (called BSB). The authentication tasks are clearly divided between the two modules – every action is only performed on one of the stations, e.g., the password-based authentication is performed by the host station without any interaction with the BSB, while biometric authentication is performed entirely by the BSB. The BSB is autonomous in its operations. The biometric database and the biometric software are placed only on the BSB, no such details are passed to the host station. Any necessary hardware must therefore be also connected to the BSB. The only information about the user passed between the two modules is a numeric identifier. The BSB is equipped with its own screen and does not need the host system as a proxy for user communication.

The architecture described in the previous paragraph is inspired by the way in which credit card terminals are often integrated with ticket vending machines, etc. The terminal is only activated on request and is autonomous. The machine instructs the user to follow the instructions displayed by the card terminal and waits for a confirmation from the terminal. This makes the integration very easy for the designers of the vending machine.

The construction of the BSB itself is a separate issue, outside the scope of this paper. If the autonomous nature of the BSB is preserved, then the issue is not at all important for the host system. The integration would be done exactly the same way if the verification used a completely different biometric modality or hardware ID. However, this elasticity depends on the flexibility of the communication protocol connecting both modules.

3. Related Work

There is a multitude of existing authentication protocols. In theory, using them would be the simplest and best way to meet the requirements of the project. However, the existing protocols are in fact designed for a completely different task. They are usually used as scalability enhancers for distributed systems, delegating the task of user authentication, authorization and accounting to a central server. This approach is used in protocols like RADIUS [9] or Diameter [10]. Also Kerberos [11], [12] uses a central server approach, focusing on a cryptographic ticket mechanism to provide single logon. All of these protocols are powerful tools enabling effective management of large networks. However, our needs are much simpler. The authentication, assignment of rights, etc., are all performed on the host system. However, the verification tasks delegated to the BSB are handled there completely.

The TACACS+ [13] protocol and its predecessors (TACACS, XTACACS) take a different approach. They do provide the means to perform remote interactive authentication, separated logically from authorization. The protocol handles password-based authentication very well and can be adapted to any other similar authentication method, e.g., hardware tokens. Adapting it to use biometric verification is a lot more difficult, but probably possible. However, the assumptions of the TACACS family of protocols are a reversed version of our design. The authentication server performs the actual authentication, but the user interaction is done on the terminal side. In our setting the entire verification process is performed on the BSB side. The host system simply requests the authorization to be performed and awaits results. This makes TACACS+ a suboptimal

[1]This is a simplification, multiple VMs may in fact use the same CPU if and *only* if they belong to the same security domain.

choice for our system. Taking into account the minimal amount of necessary communication, the decision was to develop a specialized protocol.
Since biometric verification was our main goal in this project, it was also possible to use an existing biometric protocol. Such a protocol exists – the standard biometric API called BioAPI [14] can be mapped into network messages using ASN.1, resulting in BIP – BioAPI Internetworking Protocol [15]. However, from our point of view this protocol is very low-level, giving access to many details of the biometric processing. It would be useful, if the user database was placed on the host machine, but since all biometric activity is limited to the BSB, introducing such low level information in the protocol only serves to make it less universal. A simpler protocol could easily be used with different configurations of the BSB, using different biometric modalities, or even non-biometric methods.

4. Communication Protocol

The design of a communication protocol for identity verification is relatively simple in this setting – the required set of operations is very small. In fact, the minimal set would include only two operations: adding a user by registering the necessary data given the user's identifier or verification of the identity of the user given the identifier of the user he claims to be. For practical reasons, a third operation is quite useful – deleting the user with the given identifier, removing all data collected for him.
The basic design requirements, apart from providing the specified minimal set of operations, were as follows:

1. The design must be based on well known, standard solutions.
2. The autonomous nature of BSB, as specified in the proposed architecture, must be preserved.
3. The messages should be limited to ASCII characters and not excessively long, so that the protocol can be used unmodified on any link.
4. The messages should be human readable when unencrypted (useful in implementation and testing phase).
5. The protocol should be easy to implement.
6. The protocol should be easily extendable.

The requirements and the natural request-response nature of communication with BSB resulted in a solution based on Web Services. The choice between SOAP and XML-RPC was also clear, as SOAP is unnecessarily complex – we have no use for its advanced capabilities and the large envelope directly contradicts the requirements. Since ASCII is a subset of UTF-8, it is easy to satisfy requirement 3 using XML entities and BASE64 encoding where necessary. Also, the clear hierarchical structure of XML documents satisfies requirement 6.

4.1. Extended Functionality for More Advanced BSBs

While the three basic operations are sufficient for the verification module, the actual protocol provides two additional, optional features – capabilities and log collection.
Capabilities, although simple, are a powerful extension, allowing more advanced verification modules with multiple verification methods to be designed and used without further modifications to the protocol. A capability is simply a name of a verification method. The protocol allows the host to obtain a list of capabilities supported by the verification module and to specify the desired set of methods during verification or registration of a user's identity. The verification module is also required to define internally a default set of verification mechanisms, allowing it to be used with a host which does not support capabilities.
Capabilities are described in the format `<capabilityname>_<mode>`. The `<capabilityname>` part identifies the general method of verification (e.g., biometric modality) and should be unified. The BSBI protocol provides names for most popular verification methods:

- **CHIP** Chip ID card or other intelligent hardware ID inserted into a reader,
- **FACEGM** Face geometry,
- **FPRINT** Fingerprint,
- **FVEIN** Vein pattern in a finger,
- **HANDGM** Hand geometry,
- **IRIS** Iris,
- **PASS** Password or PIN code,
- **RCHIP** Wireless chip card or other intelligent wireless hardware ID,
- **RETINA** Retina,
- **RFID** RFID-based ID card or other simple wireless hardware ID,
- **SIGN** Written signature,
- **SPASS** One-time-password,
- **SWIPE** Magnetic ID card or other simple hardware ID inserted into a reader,
- **TOKEN** Hardware token,
- **VOICE** Voice.

The above list is by no means complete. Additions to it should be done in a consistent manner.
The `<mode>` part is not strictly defined – it may be any string (without whitespace) specifying a variant of the method, preferably in a clear and descriptive way. Short mode names are preferred. If only one variant is imple-

mented for a given capability, it is allowed to use only the capability name as its identifier, but the preferred name uses mode `default`. This mode is not reserved for that use, so it is, for example, perfectly acceptable to offer capabilities `SIGN_default`, `SIGN_press` and `SIGN_nopress`, where the latter two modes define whether pressure measurements should be taken into account and the `default` mode is synonymous to one of the other variants. Synonyms are not treated in any special way, from the point of view of the protocol all modes are different.
The log collection is simply a way to access more detailed information about the operations of the verification module. The host system may use it to obtain a copy of some of the module's logs. The proposed specification assumes that the logs are treated as a single stream of entries by the host, delegating the selection of entries to the verification module. Better granularity may be provided by extending the protocol, if necessary.

4.2. Specification of Operations

The protocol consists of five methods – one for each of the elementary operations of the BSB, one listing the capabilities of the BSB and one used to collect the logs. The methods generally report their execution status using numeric error codes, which are defined as follows:

- 0 `BSBI_SUCCESS` – successful completion;
- 99 `BSBI_UNKNOWN` – unknown error;
- 100 `BSBI_USER_NEXIST` – the given user ID is not registered in BSB database;
- 101 `BSBI_USER_EXIST` – the given user ID is already registered in BSB database;
- 110 `BSBI_USER_LIMIT` – number of registered users exceeds a preset limit;
- 200 `BSBI_PROC_TIMEOUT` – processing timeout in verification module;
- 210 `BSBI_PROC_INTERNAL` – internal error in verification module;
- 299 `BSBI_PROC_UNKNOWN` – unknown error in verification module;
- 300 `BSBI_CAPA_NSUPPORT` – requested capability not available in the BSB;
- 301 `BSBI_CAPA_NCOLLECT` – data required by the requested capability not collected;
- 310 `BSBI_CAPA_NODATA` – no user template for the requested capability for the specified user.

Additionally the following library-level error codes (900–999) are defined. These are never sent as part of the protocol and are reserved for use by libraries implementing BSBI:

- 900 `BSBI_CONN_FAIL` – cannot establish connection to BSB (connection refused or other problem);
- 901 `BSBI_CONN_TIMEOUT` – timeout while waiting for BSB response;
- 910 `BSBI_CONN_PARSE` – bogus response from BSB;
- 920 `BSBI_CONN_FATAL` – BSB or link security breach suspected (e.g. invalid BSB certificate);
- 930 `BSBI_CONN_INTERNAL` – BSBI library internal error;
- 940 `BSBI_CONN_PROTO` – protocol incompatibility – BSB does not support requested functionality;
- 999 `BSBI_CONN_UNKNOWN` – unknown connection error.

The methods of the protocol are defined as follows:

enroll.user(uid, *capa*) – the method registers a new user with the supplied ID. The BSB is expected to collect and store autonomously all necessary information, e.g., biometric data.
The mandatory parameter `uid` contains the numeric ID of the user.
The optional parameter `capa` may contain a list of verification methods the host system intends to use in the future. The BSB is required to collect and store all necessary data for these verification methods, failure to collect any of them must cause the entire operation to fail. It may also collect data for other variants, but failure to obtain those must not be considered an error. Specifying a capability not supported by the BSB is an error and user enrollment must fail in this case. If the list contains a capability specified only by capability name (omitted mode part of the name), the BSB is free to use any of the available modes of that capability.
If the parameter `capa` is not provided, the BSB is free to choose which kinds of data should be collected. As a minimum, the BSB should collect data for all capabilities used by default (that is when the `capa` parameter is omitted) by the `verify.user` method – failure to collect those may and should be considered a failure to register the user. The preferred action is to collect data for all installed capabilities, but tolerate failure to obtain data for verification methods not used by default.
The method returns a single value `code`, which is a numeric error code. The following error codes are possible in this method:

- `BSBI_SUCCESS`,
- `BSBI_UNKNOWN`,
- `BSBI_USER_EXIST`,

– BSBI_USER_LIMIT,

– BSBI_PROC_TIMEOUT,

– BSBI_PROC_INTERNAL,

– BSBI_PROC_UNKNOWN,

– BSBI_CAPA_NSUPPORT (only if the capa parameter was provided),

– BSBI_CAPA_NCOLLECT.

verify.user(uid, *capa*) – the method verifies the identity of a user, based on the supplied ID. The BSB is expected to collect autonomously the necessary data and verify its correctness for that ID using templates created by the enroll.user method.
The mandatory parameter uid contains the numeric ID of the user.
The optional parameter capa may contain a list of verification methods that should be used. The verification is successful if and only if all of the listed capabilities were successfully used and confirmed the user's identity. The BSB may not use any capabilities not in the list. Specifying a capability not supported by the BSB is an error. If the list contains a capability specified only by capability name (omitted mode part of the name), the BSB is free to choose one or more of the available modes for which data was collected during enrollment of the user being verified. If user data is not available for any mode of the capability, then the verification obviously fails.
If the parameter capa is not provided, the BSB is free to choose which capabilities to use for verification and how to proceed if one of them fails. The choice should be limited to the capabilities used during enrollment of the user being verified.
The method returns one or two values. The first value is code, a numeric error code. The following error codes are possible in this method:

– BSBI_SUCCESS,

– BSBI_UNKNOWN,

– BSBI_USER_NEXIST,

– BSBI_PROC_TIMEOUT,

– BSBI_PROC_INTERNAL,

– BSBI_PROC_UNKNOWN,

– BSBI_CAPA_NSUPPORT (only if the capa parameter was provided),

– BSBI_CAPA_NODATA.

The second value, also numeric, called verificationResult, is provided if and only if the value of code is BSBI_SUCCESS and is either 0 if the user's identity was confirmed, or 1 if the user's identity was rejected.

delete.user(uid) – the method removes from the BSB all data related to a user identified by the supplied ID.
The mandatory parameter uid contains the numeric ID of the user.
The method returns a single value code, which is a numeric error code. The following error codes are possible in this method:

– BSBI_SUCCESS,

– BSBI_UNKNOWN,

– BSBI_USER_NEXIST,

– BSBI_PROC_UNKNOWN.

list.capa() – the method retrieves a list of all capabilities supported by the BSB.
The method does not require any parameters and returns an array of character strings called capa. There is no code value in this case – any non-empty response should be regarded as confirming the result BSBI_SUCCESS, while an empty response should be interpreted as BSBI_UNKNOWN.

get.logs(*zipmode*) – the method retrieves new log entries from the BSB. Selection of the entries depends on the configuration of BSB, the protocol does not provide any method of controlling it.
The optional parameter zipmode specifies the compression method which should be applied to the logs. It is a string from a well defined set of values. Currently, the defined values are ZIP, GZ and NONE, new ones may be added in the future. The values are not case sensitive. The BSB must use the specified compression method, unless it does not support it – in that case, no compression should be used (as if the value of zipmode was NONE).
The method returns one or four values. The first value is code, which is a numeric error code. The following error codes are possible in this method:

– BSBI_SUCCESS,

– BSBI_UNKNOWN,

– BSBI_PROC_INTERNAL (reserved for errors generated when parsing BSB's logs),

– BSBI_PROC_UNKNOWN.

The other three values are returned if and only if the value of code is BSBI_SUCCESS.
The value logs contains the log entries in BASE64 encoding. If the zipmode parameter was specified and not NONE, the entries may be compressed before BASE64 encoding is applied. There is a limit set on the length of this value – it may not exceed 32 kilobytes after BASE64 encoding.
The value zipmode defines whether compression has been used and is either a copy of the input parameter with the same name, or NONE if the specified compression method is not supported.

Finally, the value `hasnext` is a boolean value, true if and only if not all available entries fit under the 32 kilobyte limit. The host system should then immediately call this method again to retrieve the missing lines.

5. Prototype

The prototype implementation (as shown in Fig. 2) was completed in the C language, using a preexisting XML-RPC library. The BSBI was implemented as a portable library, easily adaptable to new uses.

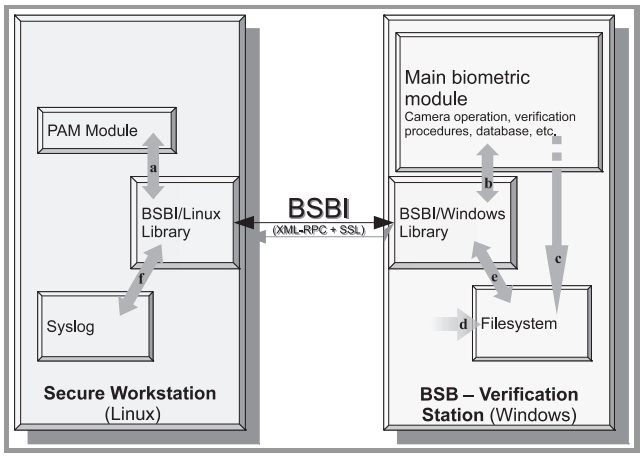

Fig. 2. The structure of the prototype implementation of the BSB: (a) verification requests are handled by a special PAM module using the BSBI library; (b) communication with the biometric module uses a simple wrapper library; (c),(d),(e),(f) events from the biometric module (c) and system-level events (d) are logged into files, which are (e) monitored by a special module of the BSBI library and (f) selected entries are reported to the host station's syslog.

The BSB has been equipped with Apache HTTP server configured to enable SSL-protected connections from the BSB. The connection requires authentication with a registered certificate by both the client and the server, ensuring a secure pairing of the BSB with the host system. The BSBI module on the server handles BSBI calls by translating them to calls to functions of a specially designed thin wrapper around the ACIrisSDK library, responsible for the actual biometric processing. The module also collects results of a separate process which parses logs generated on the BSB and extracts the entries, which will be sent to the host system on the next call to `get.logs`.

On the client side, the library is used to generate calls to BSB. It could potentially be used by regular applications requiring biometric verification of identity, but in the designed system it is reserved for use in the initial user authentication process.

Given that the host operating system in the project is Red Hat Enterprise Linux, the natural way to include the BSB in the user authentication process is to implement a module for the PAM (Pluggable Authentication Modules) system. The module is then included in PAM configuration as the second step after standard password authentication. Such a module, called `pam_biometric.so`, has been implemented. However, the most obvious authentication policy, strictly requiring successful biometric verification, is not acceptable in practice. The design of the secure workstation prohibits any external modifications (e.g., after booting from a live CD), the only way to change its configuration is to authenticate successfully as an administrator. This means that any failure of the BSB makes the host system completely unusable – even replacing the BSB is not possible, unless the original private key can be recovered.

For this reason the PAM configuration is a bit more complex. Another simple module for the PAM system was developed, called `pam_spec.so`. This module performs a simple password-based verification. However, the module only accepts users belonging to a special group `spec`, and the password is never used normally. It is also stored in a separate file, `/etc/spec_shadow`, protected using SELinux. The approach is quite secure, especially if password and BSB are not the only authentication modules used by the workstation.

The final configuration can be as follows:

```
#%PAM-1.0
auth     required     pam_unix.so nullok ↵
    try_first_pass
auth     [success=done authinfo_unavail=ignore ↵
    auth_err=die default=die]  pam_biometric.so ↵
    audit
auth     requisite    pam_succeed_if.so user ↵
    ingroup spec
auth     required     pam_spec.so audit
```

First, a successful password-based authentication is required. If it succeeds, the identity of the user is already tentatively established and a BSB-based authentication can be attempted using his identifier. As the second line specifies, successful BSB authentication is sufficient and ends the process. Rejection by BSB also ends the process – authentication fails. However, if the module reports that it is unavailable, i.e., the BSB cannot be contacted, then processing continues. The third line rejects any users not in group `spec` and the fourth line attempts special password authentication as the final, decisive step. The `audit` option used in both modules implemented in the project turns on detailed logging for audit purposes and can be omitted if not necessary.

The integration package includes both PAM modules, the BSBI library and some helper programs. The helper programs enable pairing of the host system with the BSB, testing the connection, enrollment and removal of users, setting the special passwords, etc.

6. Possible Extensions

While the BSB + BSBI verification mechanism is implemented and working well, it is still only a prototype. A fully mature commercial system would require further extensions and modifications. Some of the extensions described in this section are planned as future work, others are just proposals which may or may not be considered for implementation in the future.

Most importantly, to be used as part of a secure workstation, especially one certified for processing of classified information, the BSB would have to adhere to the same stringent security requirements as the workstation itself. Several steps in that direction were already made. The BSB, once configured, does not have any connected human interface devices and the touchscreen built into the computer used in the prototype is used only as an output device. Nevertheless, the casing is a purely temporary solution, not acceptable for production use, as it does not offer any physical security. All ports of the computer are relatively easy to access. In a final implementation the computer and the camera would be installed in a locked casing with secure, well placed ventilation holes, so that accessing any ports would require a key (it is normally only necessary during installation and initial pairing with the host). Protection of the link is also important, although as long as the BSBI communication is cryptographically protected this may not be crucial. Also note that the BSB is not really designed for use in the field – processing of classified information typically takes place in rooms providing significant physical security and well documented access control. In other applications the BSB would likely not be a separate piece of equipment. Whether wall-mounted or built into a larger device, it would probably be sufficiently protected. Even now, physical integration with the host workstation is perfectly possible if the workstation's casing is large enough. However, in the envisioned application a smaller, standalone verification device connected to the host with a cable seems much more usable as it offers a lot more flexibility in placing the camera so that using it would not require leaving the chair.

The security requirements, especially at higher levels, may require replacing SSL-based encryption and symmetric authentication with encryption hardware. This is however easy to do and does not require any changes in the BSBI protocol.

Another security-related shortcoming of the current solution is the relatively low security of the BSB itself. As long as there are no input devices connected to it (apart from the camera, of course), this is not a serious problem. However, the computer used in the BSB is small enough to carry in an average bag. The consequences of stealing the BSB or making a copy of its data are hard to define. The BSB does not have any useful metadata associated with the biometric (or other) verification data stored on it. The identifier passed to it by BSBI is not easily identifiable. However, the data may be quite useful anyway, especially if the set of registered users is sufficiently small. The connection between a person and its verification data can be recovered in several ways, e.g., by analysing the logs present on the BSB and comparing them with observations of the times when individual users accessed the workstation. The verification data, especially biometric templates, should definitely be considered sensitive.

Another interesting piece of information on the BSB is its private key, used to pair with the host. Having this certificate, it is generally possible to develop a fake BSB, which will pair correctly with the host, but will, e.g., always reply to `verify.user` calls with a positive verification. It may be possible to use the Trusted Platform Module (TPM) of the BSB to protect the key. This method, however, will not suffice to protect the verification data.

One method worth trying would be to put the biometric software along with the database on an encrypted volume. The key necessary to access that volume would be stored on external storage during the setup phase, and afterwards it could be put on the host system, preferably encrypted using the host's TPM. The BSBI protocol would then require two extensions. One new error code, called, e.g. `BSBI_NOT_READY` would be used to inform the host in reply to any call that the BSB's verification modules are not yet running. Then, a separate method of the BSBI protocol (e.g. `init`) would be used to send the necessary key to the BSB, which would then be able to start the verification services. This way the sensitive data could not be accessed without connection to the right host.

Another possible extension would be to enrich the `get.logs` method with the possibility to specify the requested type of logs, or even to implement two-way communication, where the logs are simply pushed to the host machine. The latter variant is easiest to do using syslog instead of extending BSBI.

7. Conclusions

The architecture and protocol described in this paper have been implemented and tested as part of the project "Secure workstation for special applications", but the solution is more general, not limited to this one application. We have shown that the approach borrowed from credit card terminals – making the verification station a separate, autonomous module and developing the main system as agnostic of the inner workings of that module – is indeed workable for this application. The approach may be used whenever advanced identity verification is necessary in a system in which such solutions are not readily available or may not be possible to implement due to, e.g., lack of sufficient computing power.

Acknowledgements

This work is part of the project called "Secure workstation for special applications" and is funded by a grant number OR00014011 from the National Center for Research and Development – science funding for years 2010–2012.

References

[1] A. Kozakiewicz, A. Felkner, J. Furtak, Z. Zieliński, M. Brudka, and M. Małowidzki, "Secure workstation for special applications", in *Secure and Trust Computing, Data Management, and Applications*, C. Lee, J.-M. Seigneur, J. J. Park, R. R. Wagner, Eds., Communications in Computer and Information Science, vol. 187. Berlin: Springer, 2011, pp. 174-181.

[2] L. O'Gorman, "Comparing passwords, tokens, and biometrics for user authentication", *Proceedings of the IEEE*, vol. 91, no. 12, pp. 2021–2040, 2003.

[3] H. K. Lu and A. Ali, "Communication Security between a Computer and a Hardware Token", in *Proc. Third Int. Conf. Sys. ICONS 2008*, Cancun, Mexico, 2008, pp. 220–225.

[4] R. Molva and G. Tsudik, "Authentication method with impersonal token cards", in *Proc. IEEE Comp. Soc. Symp. Res. Secur. Priv.*, Oakland, CA, USA, 1993, pp. 56–65.

[5] R. M. Bolle, J. H. Connell, S. Pankanti, N. K. Ratha, and A. W. Senior, *Guide to Biometrics*. New York: Springer, 2004.

[6] K. Ślot, *Wybrane zagadnienia Biometrii*. Warszawa: Wydawnictwo Komunikacji i Łączności, 2008 (in Polish).

[7] A. Czajka and A. Pacut, "Iris recognition system based on Zak-Gabor wavelet packets", *J. Telecom. Inform. Technol.*, no. 4, pp. 10–18, 2010.

[8] A. Czajka and A. Pacut, "Iris recognition with adaptive coding", in *Rough Sets and Knowledge Technology*, Lecture Notes in Artificial Intelligence, vol. 4481. Berlin: Springer, 2007, pp. 195–202.

[9] J. Hassell, Radius – Securing Public Access To Private Resources. O'Reilly & Associates, 2002.

[10] P. R. Calhoun, G. Zorn and P. Pan, "DIAMETER Framework Document", IETF, 2002.

[11] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks", *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 33–38, 1994.

[12] J. T. Kohl, B. C. Neuman, and T. Y. T'so, "The Evolution of the Kerberos Authentication System", in *Distributed Open Systems*, D. Johansen and F. M. T. Brazier, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1994, pp. 78–94.

[13] D. Carrell, "The TACACS+ Protocol Version 1.78", Network Working Group INTERNET-DRAFT, Cisco Systems, 1997.

[14] "Information Technology – BioAPI – Biometric Application Programming Interface – Part 1: BioAPI Specification", ISO/IEC 19784-1.

[15] "Information Technology – BioAPI Interworking Protocol (BIP)", ISO/IEC 24708.

Tomasz Pałka graduated from the Faculty of Mechatronics of Warsaw University of Technology, Poland. Currently he works as a Specialist at Systems and Information Security Methods Team in NASK Research Division. His present areas of interest are centered around the security of information systems.

E-mail: tomasz.palka@nask.pl
Research and Academic Computer Network (NASK)
Wąwozowa st 18
02-796 Warszawa, Poland

Adam Kozakiewicz – for biography, see this issue, p. 21.