Digital time stamping system based on open source technologies

R. Miškinis, D. Smirnov, E. Urba, A. Burokas
Time and Frequency Standard Laboratory
Semiconductor Physics Institute
Vilnius, Lithuania
miskinis@pfi.lt

B. Malyško
State Tax Inspectorate
Vilnius, Lithuania

P. Laud
Cybernetica AS
Estonia

F. Zuliani
Nergal S.r.l.
Italy

***Abstract*— A digital time stamping system (the BALTICTIME system) was developed by using the open source technologies and was adopted to meet the eGovernmental applications, where legal and accountable time stamps are essential. All components of the BALTICTIME system were developed by using the open source software technologies LINUX-UBUNTU, OpenTSA, OpenSSL, MySQL and are designed to suit the hardware, which is common to the national time metrology laboratories. The main technological tasks of BALTICTIME are to develop the interface between national time standard authorities, keepers of the national coordinated time scales UTC(k), and time stamping authorities (TSAs); to develop the issued time stamps archiving system integral with UTC(k) time scale; as well as to develop security system based on interoperable certificate data base. Technical specifications will be assessed according to requirements of the international standards. Test results demonstrate the BALTICTIME system's ability to issue up to 10 accountable (archived) digital time stamps per second, which fulfills the needs of eGovernment and eCommercial services. Calibration and testing procedures of time stamping units were developed.**

I. Introduction

The open source technology is the backbone of the *BALTICTIME* system, which is designed to enhance the confidence of time stamping service through adopting capacities of time standard authorities of European metrology system as the most authoritative and reliable backbone for a time stamping authority. In such a way, level of the confidence and accountability of the service eligible for employment in eGovernment services will be created and integral for Europe solution having strong potential to overcome existing fragmentation will be introduced.

The overall objective of the *BALTICTIME* project, which is under implementation from 2006, is to develop the legal and accountable digital time stamping (DTS) system providing the layer of trust in eGovernmental transaction environment and to demonstrate DTS system performance for time-critical functions or validation data for digital signature.

The main technological tasks of *BALTICTIME* are:

1. To develop the interface between the national time standard authorities, keepers of national coordinated time scales UTC(k), and the time stamping authorities;
2. To develop eDocuments archiving system integral with time scale generator traceable with the universal coordinated time scale (UTC);
3. To develop security system based on interoperable certificate data base;
4. To demonstrate *BALTICTIME* system performance for eGovernment services with a cross-border time stamping possibilities.

All components of the *BALTICTIME* system are developed using open source software technologies LINUX UBUNTU distribution and fitted for the hardware common for the National Time Metrology laboratories. Technical specifications will be assessed according to requirements stated in the standards [1 – 4].

II. Architecture of the BALTICTIME system

To cope with a large number of time stamping requests, a TSA has been deployed in a distributed manner (Fig. 1), with one of the sites acting as a main system (TSA main site), keeping its subsystems (Time

978-1-4244-3510-4/09/$25.00 ©2009 IEEE

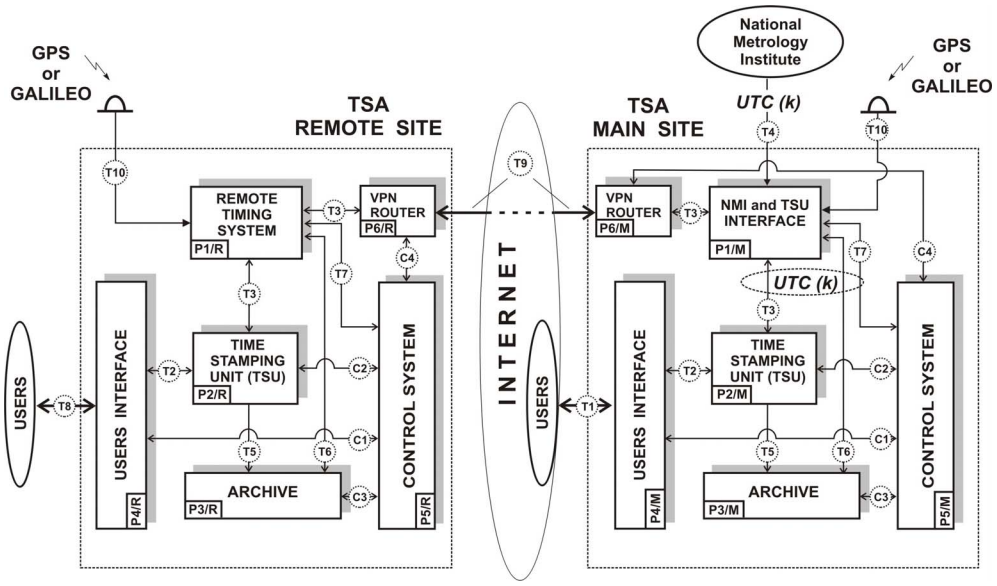

Fig. 1. Architecture of the BALTICTIME system. UTC(k) is the universal coordinated time scale of the "k" national metrology institute.

Stamping Units (TSU), Archive Units (AU) and others) synchronized with the Universal Coordinated Time Scale (directly connected to the National Metrology Institute (NMI)), while the other sites (TSA remote sites) synchronized and linked with the main site. A distributed system can also provide better availability which is important in time stamping applications where some party is always interested in obtaining the timestamp as soon as possible. The hardware configuration of the BALTICTIME is shown in Fig. 2.

III. TIMING SUBSYSTEM

The timing subsystem generates local TSA timescale, keeps it synchronized to UTC(k) generated by National Metrology Institute and provides it to the all components of main and remote TSA sites. The key parts of time mark synchronization system are:

1. TSA timing Subsystem (located in TSA Main Site);
2. NMI timing Subsystem (located in NMI);
3. Remote Timing Subsystem (located in TSA Remote Site).

Time stamping authorities will issue legal time stamps (TS), *i. e.,* TS should be traceable to UTC(k). The main purpose of TSA's timing subsystem is to generate TSA's local time scale synchronized to UTC(k) – time scale generated by the national metrology institute (NMI) and provide it to the time stamping units (TSU) of the time stamping authority. In order to provide system redundancy TSA's timing subsystem should be equipped with at least 3 time standards. Frequency stability of time standards depends on required time stamping resolution. The system after initial synchronization must work autonomously at least for 10 days without external time comparisons to NMI.

1PPS signal from time standards should be connected through multiplexer and time interval counter to Timing Subsystem's Control Unit and compared continuously. Control Unit should be able to detect improper results from one of time standards, and eliminate the bad-working time standard. In case when less than 2 time standards work correctly control unit should stop the time stamping process.

In order to provide required accuracy, and for security reasons, trusted TSA must be synchronized to NMI using at least two independent methods. Use of GNSS based time transfer as a primary method, and NTPv4 as secondary method is recommended. Required accuracy of time transfer methods depends from required time stamping resolution.

IV. TIME STAMPING UNITS

Each of the BALTICTIME's time stamping units (TSU) implements an RFC3161-compliant timestamping server, also including the security-enhancing extensions specific to BALTICTIME. The implementation is based on the open-source cryptographic library OpenSSL, and

This project was sponsored by the European Commission under the programme FP6.

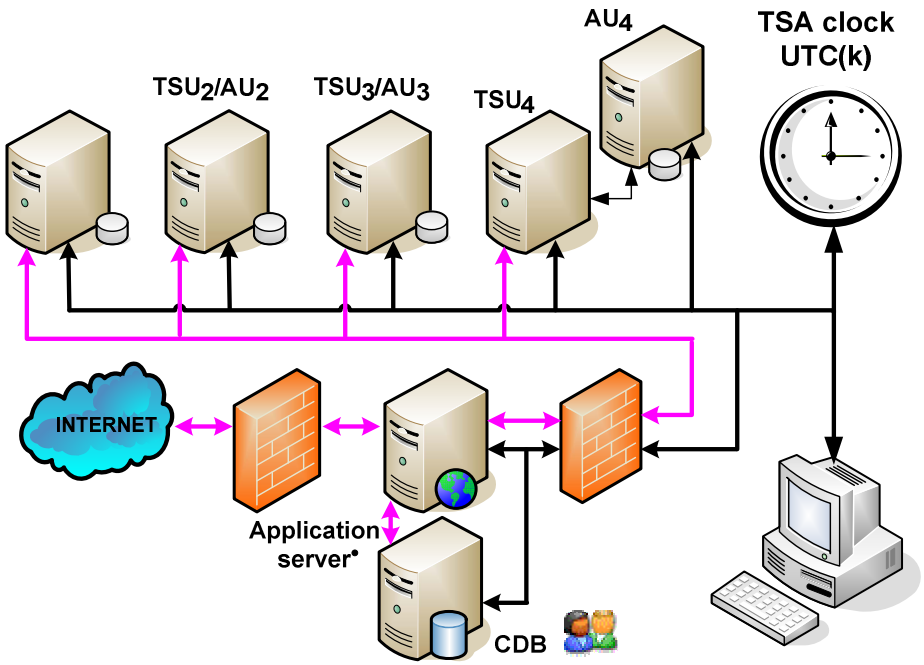

Fig. 2. The hardware configuration of the BALTICTIME.

in particular its extension OpenTSA implementing the functionality described in RFC3161. OpenTSA implements a typical hash-and-sign timestamping server where the document digest contained in a client's query is signed by the timestamping server, using a dedicated signing key. Key management of the TSU is done according to the standard [ETSI TS 102 023 v1.2.1]. The certificates for the public key are distributed in the same way as certificates for signing keys of any other party. In such a timestamping system, the leakage of the server's signing key would mean the invalidation of all timestamps issued with this key, unless the signing time of those timestamps can be determined using other means. To protect against such leakage, and to also prevent a malicious timestamping server from reordering the timestamps, all timestamps issued by the same TSU are submitted to the archive unit (AU) that will link them together using a one-way hash function (refer to the description of the AU for a more thorough description of linking). The current state of the linking chain (called the *last linking item*, and having the size comparable to the size of the output of the hash function) has to be periodically included in the timestamps to complete the chain of one-way

dependencies between different timestamps. Hence the TSU will periodically (preferably before each time it issues a timestamp) query the AU for the last linking item. This item is included in the timestamps as a signed attribute (RFC3161 contains provisions for this). Tha details of such inclusion were worked out in the scope of Baltictime project. The implementation of the TSU consists of the necessary modifications to OpenTSA to construct timestamps also containing the last linking item (implemented in C), as well as a small program communicating with the client and the AU, implementing the necessary protocol logic, and using the modified OpenTSA-library (implemented in OCaml).

V. ARCHIVAL UNITS

An archival unit keeps a log of timestamping server's activity, thereby allowing to audit it and to compare the timestamps after the keys used to sign them have expired. The archival unit achieves this objective by linking the timestamps together using a one-way cryptographic hash function.

Since a TSA may have multiple TSUs, and hence multiple sites, the audit mechanism must take into account that two timestamps might have been issued by different TSUs. In order to create a single arrow of time between two timestamps issued by two different TSUs the archival units periodically send their last log item to

all of the other archival units in order for them to include it in their linking items chain. In this way it is possible to create a linking chain between two timestamps stored in two different archival units.

A cryptographically secure hash function *h* has been fixed. It is used to link the log items together. The hash function is not hardcoded, but the archival unit stores its identity. If necessary, the hash function may be changed (although this is supposed to be an infrequent and always an extraordinary event). The chosen hash-function must be considered collision-resistant for the foreseeable future. The log is a set of items of the form (n, X_n, L_n), where X_n is the n-th bitstring (timestamp) saved in the log and $L_n = h(X_n, L_{n-1})$ is the linking information. The log items may also need to store the source of the bit-string X_n (which may be the TSU, or the time mark synchronization unit, or the control system, or some remote archival unit) and the recipient(s) of L_n, if there are any (they may be the TSU or some remote archival units).

The configuration of the archival unit must include the addresses of other archival units, with whom the linking information is exchanged. For each archival unit we have to store its address (probably an IP address and a port number) and the frequency of synchronization.

The main functionalities of the TSA's archive system are:

to store issued timestamp and timestamps linkage information;

to send periodically the last link item information to other archives in order to link them.

The Archival Unit consists of a database and software performing the required accesses to the database and the network communications with other modules such as the TSU and the other remote archive units. The Archival Unit relies on a relational database system to store its data. Since the usage of open source technologies was strongly suggested, the chosen database system is MySQL. The application interaction to the MySQL database is carried out using the MySQL++ library. This library is a C++ wrapper of the standard MySQL development libraries to interact with MySQL database (http://tangentsoft.net/mysql++/). This is referred to as the official C++ opensource wrapper of the MySQL connector ANSI C driver. The communication between Archive Unit and other connected modules is carried out through network protocols in a TCP socket connection. The archival unit works as a server responding to requests that other modules perform. Any TCP/IP network connections and communications is carried out using the library "C++ Sockets" (http://www.alhem.net/Sockets/). In order to encode and decode all the messages that the archival unit exchanges with the other modules, an open-source external package that handles ASN1 has been used (http://lionet.info/asn1c). The generated code has been included in the archival unit software as a static library. Data hashing and base64 encoding is achieved using the open-source OpenSSL library (http://www.openssl.org). Implemented scheme of archival units linking has not attempted before [6].

VI. USER'S INTERFACE

The TSA main site, which configuration is presented in Fig. 2, serves the needs of small users. For the connection between the small user and the TSA the WebServices are used, the user will have a possibility to write his own client application by himself, using any programming environment and language. The necessary information to do that is be available in the WSDL (Web Service Description Language) file and additional documentation is also available at the TSA internet site. As an alternative, there are premade user Java applets published on TSA web site. Those applets provide all needed functionality for time stamp generation and time stamp linking chain verification. The only stipulation for client applet usage is that the user must have installed on his/her PC the JAVA run-time environment.

To be able to use the TS service, a small user will has to register. The registration procedure is simplified for project purposes; the user must only provide username, password and user security certificate. To ensure the security, transparency and reliability of the service, there is no application for the generation of the key pair (private and public keys). The user has to generate the key pair manually with JAVA KeyTool (the detailed instructions is available in the site). JAVA KeyStore isused for the storage of user certificate and private key. For the project purposes, self signed certificates are allowed. In the final version of the TSA only trusted CA's will be accepted and CRL (Certificate Revocation List) checked. The user will have to enter his username, select the private key from the password-protected KeyStore, and then browse for the file to be time-stamped. The user will be able to choose between several hash algorithms to be used for file hash calculation. The following data will be sent to the server: the username, hash, hash algorithm and a signature calculated from hash string with the user private key. After the submission of the data, the username and certificate is used by the TSA to identify the user. TSA will use the user's public key (from the Certificate submitted at registration) to verify the sender's identity and request integrity as well as to authorize the provision of the TS service. Applet sends the above mentioned data set to the TSA's WebService

using HTTPS connection. All data are sent using the SOAP protocol message (look in Appendix). Application server will create ASN1 encoded request to the TSU and will then send it to TSU server through socket connection (TSU server selection is load balanced).

Response is also in SOAP format and consist the Base64 encoded string representing the encoded bytes of the TimeStamp.

VII. TEST RESULTS

The BALTICTIME system has undergone extensive testing, with several types of tests performed. The results we describe here are the dependencies of the BALTICTIME's behaviour (the mean response time and the rate of failures) on the conditions of normal operation – time interval between requests and the number of clients requesting a time stamp at the same time (such clients are further called concurrent clients).

Notes: 1) direct operation mode implies that the requests are sent to an online TSU (a single device), while the operation via the application server (APP) implies that, unless said otherwise, the requests are sent by the internet and distributed by the APP to the four TSUs; 2) failure in this context means that some of the requests result in Time out instead of a time stamp returned. In no means does it imply the collapse of the BALTICTIME; 3) all the requests were sent from Vilnius, Lithuania; in all the cases (except for the the cross-border test) the TSA situated at the Semiconductor Physics Institute was interrogated.

- In the direct operation mode, the mean delay in the response to consequent requests (i.e., when a new request is issued after the response to the previous one has been received) of a single client is 17.14 ms. If there are 10 concurrent clients, the mean delay is 95.8 ms.
- In the operation via the application server (APP), the mean response delay is 82.89 ms for a single client and 133.9 ms for the case of 10 concurrent clients.

Now, let's consider the case when a new request is sent after a certain time from the previous request elapses, regardless of whether the response to the previous request has come or no. It may be expected that with the time interval between the requests decreasing, a limit shall be reached where the BALTICTIME is unable to cope with too frequent requests, which shall result in failures. Whereas the response delay can be expected to increase. Fig. 3 shows the mean time stamp response delay versus the time interval between subsequent requests in the mode of direct operation – in the case of a single client and two concurrent clients. It is seen that when the time interval between requests is larger than the "normal" response delay, the response delay is actually independent on the time interval between requests. However, with the request rate increasing, the response

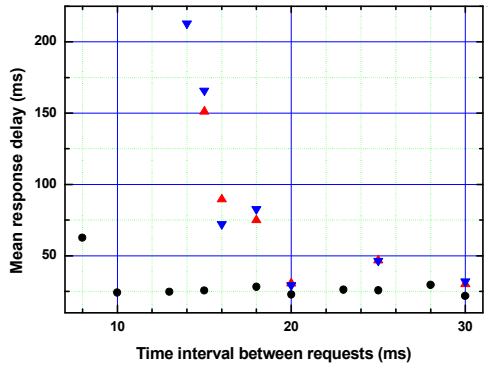

Fig. 3. Mean time stamp response delay versus the time interval between subsequent requests in the mode of direct operation: the case of a single client (black circles) and two concurrent clients (blue and red triangles).

delay increases too. It is to be noted that in the single client case failures occur when the time interval between subsequent requests < 10 ms, while in the case of two concurrent clients – when the time interval between subsequent requests < 15 ms.

Fig. 4 shows the same results obtained while operating the BALTICTIME via the application server. It is to be noted that in the single client case failures occur when the time interval between subsequent requests < 10 ms, while in the case of two concurrent clients, if time

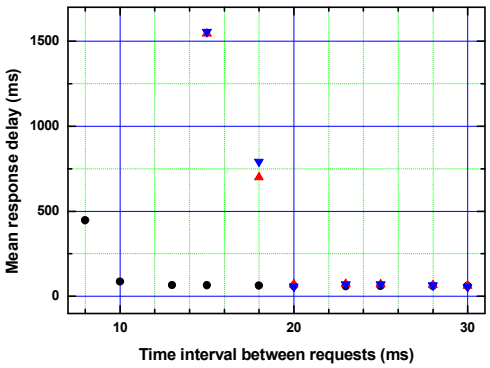

Fig. 4. Mean time stamp response delay versus the time interval between subsequent requests sent via the application server: the case of a single client (black circles) and two concurrent clients (blue and red triangles).

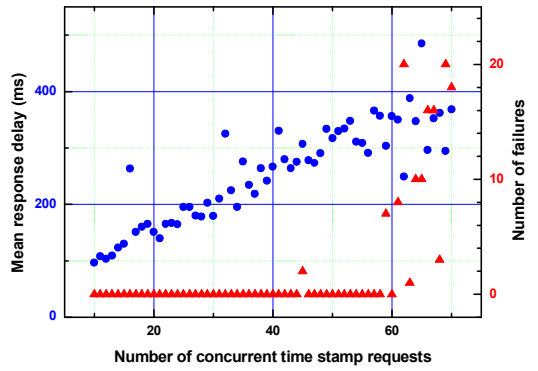

Fig. 5. Direct operation mode. Mean time stamp response delay (blue circles) and number of failures (red triangles) versus the number of concurrent time stamp requests.

interval between subsequent requests ≤10 ms, the mean response delay exceeds 1 s, which is unacceptable according to the specification.

Further, we will consider BALTICTIME's response to many simultaneous, or concurrent, requests sent only once. It may be expected that with the number of concurrent requests increasing, a limit shall be reached where the BALTICTIME is unable to cope with too

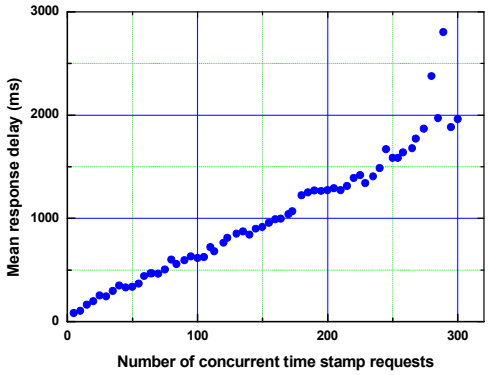

Fig. 6. Operation via the application server. Mean time stamp response delay versus the number of concurrent time stamp requests.

many requests, which shall result in failures. Whereas the response delay can be expected to increase. Fig. 5 reveals the mean time stamp response delay and number of failures versus the number of concurrent time stamp requests in the direct operation mode. It is seen that failures are unlikely if the number of requests <45. Fig. 6 shows the mean time stamp response delay versus the

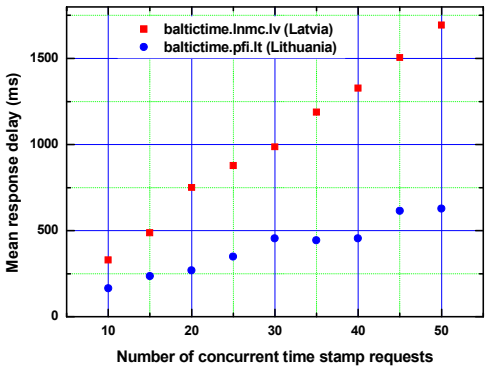

Fig. 7. Operation via the application server, with a single TSU available. Mean delay of the time stamp responses obtained from the Lithuanian TSA (blue) and the Latvian TSA (red) versus the number of concurrent time stamp requests.

number of concurrent time stamp requests when the requests are sent via the application server. It is seen that if the number of requests > 150, the delay ≥ 1 s

Finally, consider a cross-border test. To this end, we have performed a test similar to that with results shown in Fig. 6, except that there was only one on-line TSU available at our Lithuanian TSA instead of the four. Then, we performed the same sending requests to the TSA in Latvia (i.e., abroad), operating via the APP in the same conditions – with a single on-line TSU. Fig. 7 shows the results which imply that the delays in responses received from the local and abroad TSA are close to proportional.

REFERENCES

[1] ETSI TS 101 733 v1.5.1 (2003-12) "Electronic signatures and infrastructure (ESI); electronic signature formats".

[2] ETSI TS 101 861 v1.2.1 (2002-03) "Time stamping profile".

[3] ETSI TS 102 023 v1.2.1 (2003-01) "Electronic signatures and infrastructure (ESI); Policy requirements for time-stamping authorities".

[4] RFC 3161 (2001) "Internet X.509 public key infrastructure time – stamp protocol (TSP)".

[5] S.Haber, W. Scott Stornetta, "How to time-stamp a digital document", J. Cryptology, vol. 3(2), pp. 99-111, 1991.

[6] A.Ansper, A.Buldas, M.Saarepera, J.Willemson, "Improving the availability of time-stamping services. information security and privacy", 6th Australasian Conference, ACISP 2001, pp. 360-375.