The use of checksums to ensure data integrity in the healthcare industry

Carey G. Smoak[1], Mario Widel[1], Sy Truong[2]

[1]Roche Molecular Systems, Inc., Pleasanton, CA, USA, [2]MetaXceed, Inc. (MXI), Fremont, CA, USA

Want to ensure data integrity? Want to know if a file has been altered? Checksum, please! Checksums have a variety of applications, such as verifying that an application has been installed correctly and providing a method of verifying whether or not a file has been altered. For example, checksums can be used in a SAS® program to verify that a comma delimited (comma-separated values, CSV) file has not been altered before importing the CSV file into a SAS dataset. Fortunately, all operating systems have some utility available to do checksums. This paper will provide some background on the use of checksums and concentrate on a particular example using lab instrument data.

Keywords: Checksum, Data, Integrity

Introduction

The history of secretly transmitting messages (e.g. cryptography, ciphers) is an idea which has its roots in antiquity.[1] While secretly transmitting messages is not necessarily related to the computer age, it demonstrates that securely transmitting messages has been thought about for millennia. Checksums may be thought of as a special type of cryptography. The technical definition of a checksum is a count of the number of bits in a transmission unit. A checksum is a fixed size number computed from a block of digital data by using a process called 'hash function', 'checksum function', or 'checksum algorithm' that enables the detection of transmission errors.

Correctly transmitting data is a challenge in the internet age. Data that are considered correctly transmitted are when its contents are transferred and when it is received; that both are identical. We have two cases to consider when correctness is potentially compromised:

1. erratic or random transmission errors of transmitted files — checksums are good for detecting these types of modifications;
2. intentional modification of transmitted files — checksums are not so good for detecting this type of modification.

For example, suppose that one wants to use a SAS program to import ASCII files into SAS datasets. In this case, the ASCII files were downloaded from an FTP server. The transmission could introduce corruption to the ASCII files. In the second example, another user could have modified the ASCII files before the use of the SAS program. For example, the intended ASCII file has been modified by someone with good intention of making the column headers more clear. This modification may or may not be malicious, but the transmission is no longer identical to the source which makes it incorrect.

This paper will focus on the error detection of data during transmission. If we must know that a message has been accidentally modified in any way, we must use a checksum. For example, the MD5 checksum of the previous sentence is displayed below in hexadecimal format:

86fa7014d5c52f166c907d3777d30c61

This checksum was obtained on a Windows PC using the FCIV function which is available from Microsoft (http://www.microsoft.com/en-us/download/details.aspx?id=11533&mnui=4). As you can see from the example above, a checksum is typically a number obtained by using a checksum procedure similar to the FCIV function. Checksums are also referred to as hash functions or fingerprints.

To run FCIV on a Windows PC:

- download FCIV from the Microsoft website;
- put the file (fciv.exe) in a folder (e.g. C:\temp);
- open a command prompt;
- navigate to the folder where you put fciv (e.g. C:\temp);
- if the file (e.g. demog.csv) that you want to run the checksum on is in the same folder as fciv.exe, then issue the following command at the command line:
  - fciv test.csv;
- if the file that you want to run the checksum on is not in the same folder as fciv.exe, then simply add the location of the file.

Correspondence to: C G Smoak, Roche Molecular Systems, Inc., 4300 Hacienda Drive, Pleasanton, CA 94588, USA. Email: carey.smoak@roche.com

Published by Maney
DOI 10.1179/1757092112Z.0000000006

Here is a simple example of running the FCIV function within a SAS program on a comma-separated values (CSV) file using the SHA1 algorithm:

```
options nodate ls=128 ps=40 pageno=1
orientation=landscape;
data _null_;
    call system("command/c u:\fciv\fciv.
    exe u:\fciv\demog.csv -sha1 > u:
    \fciv\list.txt");
run;
```

This code runs the FCIV function (fciv.exe) on a CSV file (demog.csv) and requests that FCIV use the SHA1 algorithm for calculating the checksum. Finally, the checksum is written to a file called list.txt.

Uses of Checksums in Healthcare Industry

There are a variety of uses of checksums within the healthcare, including eCTD submissions to the FDA, SAS/Software installation, migrating files from one server to another, data transfers, and handling electronic files generated by lab instruments. Each of these scenarios will be discussed in this paper.

eCTD

Checksums should be provided for each file in an eCTD (electronic Common Technical Document) submissions to the FDA. The checksums allow the end-user to verify that the files were not altered during transmission to the FDA.[2,3] The verification that the files have not been altered is done by the end-user comparing their own checksum for each file with the checksum provided by the initial user. If the two checksums agree, then the end-user can be confident that the files have not been altered.

Software installation

An example of software installation is the installation of SAS on a server. SAS provides checksums and installation/operational qualification programs to verify that SAS has been installed and operates correctly. The user can run the installation/operational qualification (IQ/OQ) programs provided by SAS. The resulting output (a PDF file) will show the checksums provided by SAS and those generated when installing the product agree.[4,5] Thus, the user installing SAS can easily verify that SAS has been installed and operates correctly. SAS also provides notes identifying known instances where the checksum will not agree.[6] Thus, the user will be prompted to then verify these failures (when checksums disagree) using another method, or explain the failure in their documentation. This verification process of comparing checksums can be used with software products other than SAS.[7]

File migration (server)

Another use of checksums is verifying the migration of files from one server to another.[5] Checksums are performed for each file being migrated from the old server. If the two checksums for each file agree, then the user can be confident that the files have not been altered during the migration. This is not a simple task. Two of the authors on this paper were involved in such a project. It took 22 hours to migrate all of the files and 13 hours to compare all of the checksums.[5]

Data transfers

Checksums are also useful in verifying that data transfers have been performed correctly, e.g. the file has not been corrupted during transmission of the file.[8–12] Thus, the integrity of data transfers can be verified using checksums.

Instrument data

A special case of data transfers is described in this paper. Data from a laboratory instrument stored on a hard drive can be exported as an Extensible Markup Language (XML) file. When the XML file is generated, a checksum accompanies it. A validated tool can then convert data in the XML file to a CSV file. The validated tool first compares the checksum of the XML file with its own checksum on the same file. If the two checksums agree, then the tool parses data from the XML file into a CSV file. When the CSV file is generated, a new checksum accompanies it.

Then a SAS programs is run to import the CSV file into a SAS dataset. First, the SAS program generates a checksum on the CSV file and compares it to the checksum generated by the validated tool that also generated the CSV file. If the two checksums agree, then the SAS program creates a SAS dataset from the CSV file.

Thus, the user can be confident that the data in the SAS dataset are the same as the original data on the hard drive of the laboratory instrument. At each step in the process, checksums are used to verify data integrity. This process is illustrated in Fig. 1.

Password protection

Passwords are not only used in the healthcare industry, but in our everyday transactions. A recent article in Consumer Reports discusses the topic of hack-proofing passwords.[13] The article points out that passwords are not stored, rather their checksum. This is to make the passwords more secure because the original password is never written into the system and the checksum itself does not allow you to find the original password. However, certain types of checksums are more secure or robust than others — see article by Wang and Yu for more details.[14] For example, SHA1 (Secure Hash Algorithm 1) came as a response to theoretical attacks on MD5 in the 1990s. Further along the road, SHA1 showed some vulnerabilities and SHA2 (which consists of a family of four

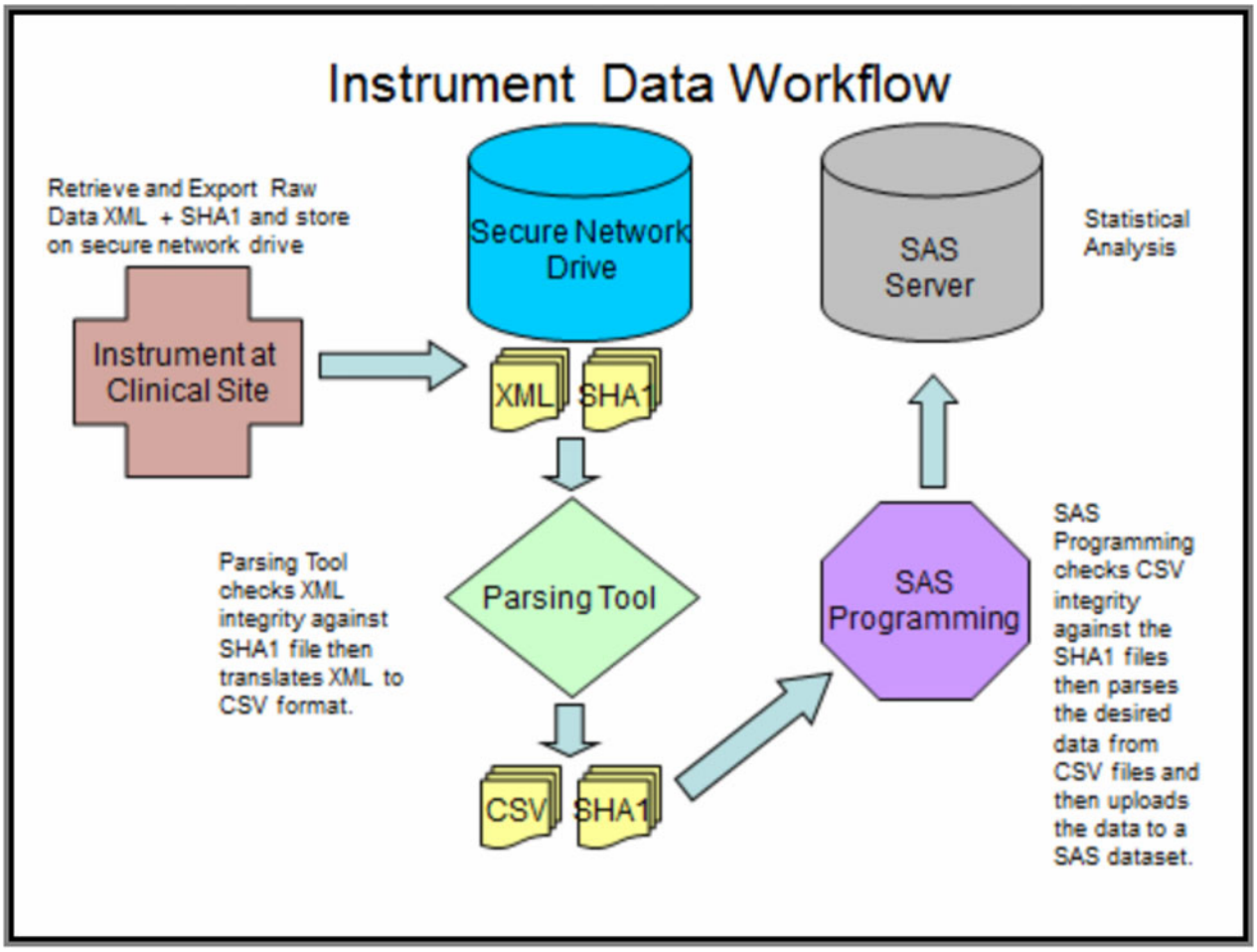

Figure 1 Data workflow

messages digests: SHA224, SHA256, SHA384, and SHA512) were developed. More recently, NSA started developing an even more secured version: SHA3.

How are passwords hacked? Very simple, you just need to find any string that will replicate the checksum of the password that you want to hack! Of course, you need to know the checksum of the password and you need to know what checksum type (MD5 or SHA1) has been applied to the password. Any public checksum utility can be used to compute checksums of arbitrary strings. After a few billion tries, you might be successful in hacking a password!

Another approach to hacking passwords is using a dictionary where instead of trying billions of arbitrary strings, you use common words from the dictionary. Thus, special characters and numbers imbedded in passwords protect against this kind of attack. However, we should not get carried away with creating passwords which are so convoluted that they are hard to type or difficult to remember. Users who write down (e.g. on sticky notes and place them next to their computer) convoluted passwords, defeat the purpose of having a password at all.

Users who want more information on how the algorithms for checksums are computed should see the paper by Rivest.[15]

Conclusion

The most significant vulnerabilities in maintaining data integrity occurs when it is being transferred. This can happen when the binaries of a software is being installed from the intended vendor to a user's machine, or when lab data in CSV format are transferred to SAS datasets for analysis or when a clinical trial is submitted to the FDA for approval. These examples highlight an inflection point in the integrity of the data as they are transferred from its original form, which is correct, to a new environment that introduce unknown changes to the data. You can ensure that the data are correct by comparing them from their original source to their final transformed location. This paper illustrates the use of checksum which provide an effective way of performing this comparison and confirms if the data's integrity is intact. The SAS data used in the examples of this paper illustrate how varied and complex transformations to the original data can be. Regardless of the different environments that the SAS system and its data are transformed, the use of checksum can be applied to consistently verify its correctness and integrity. The core concepts of checksum have roots in cryptography that dates back hundreds of years. This is a testament to how

checksum is relevant and effective in today's SAS implementations as it stands up over time.

References

1 Smoak C, Widel M, Truong S. Checksum please: a way to ensure data integrity. In: Proceedings of the Pharmaceutical SAS Users Group; 2012 May 13–16; San Francisco, CA, USA; Cary (NC): SAS Institute; 2012. Paper TA01.
2 Wang S. eCTD — a new standard for FDA electronic submission. In: Proceedings of the Pharmaceutical SAS Users Group; 2004 May 23–26; San Diego, CA, USA; Cary (NC): SAS Institute; 2004. Paper FC01.
3 Bairnsfather S. Progress toward standardization of submissions with the electronic common technical document and the evolving standardization of clinical data. In: Proceedings of the Pharmaceutical SAS Users Group; 2003 May 4–7, Miami, FL, USA; Cary (NC): SAS Institute; 2003.
4 Helton E, McNealy J, Halley, PB. SAS® validation in the pharmaceutical industry. In: Proceedings of the Pharmaceutical SAS Users Group; 2003 May 4–7, Miami, FL, USA; Cary (NC): SAS Institute; 2003.
5 Truong S, Smoak C. Validating and migrating SAS® 9.13 to 9.2. In: Proceedings of the Western Users of SAS Software; 2010 Nov 3–5; San Diego, CA, USA; Cary (NC): SAS Institute; 2010.
6 Csont WC, Proskin HM. Taking a walk on the wildside: use of the PROC CDISC-SDTM 3.1 format. In: Proceedings of the Pharmaceutical SAS Users Group; 2007 Jun 3–6; Denver, CO, USA; Cary (NC): SAS Institute; 2007.
7 Baucom J. Pharmacogenomics: JMP right in! In: Proceedings of the Pharmaceutical SAS Users Group; 2009 Mar 22–25; Washington, DC, USA. Cary (NC): SAS Institute; 2009.
8 Grasse D, Nelson GS. Base SAS vs. data integration studio: understanding ETL and the SAS tools used to support it. In: Proceedings of the 31st SAS Users Group International; 2006 Mar 26–29; San Francisco, CA, USA; Cary (NC): SAS Institute; 2006. Paper 099-31.
9 Mehler G. Next generation data warehousing with SAS 9. In: Proceedings of the 29th SAS Users Group International; 2004 May 9–12; Montreal, Canada; Cary (NC): SAS Institute; 2004. Paper 115-29.
10 Rausch N. Stars and models: how to build and maintain star schemas using SAS® data integration server in SAS®9. In: Proceedings of the 31st SAS Users Group International; 2006 Mar 26–29; San Francisco, CA, USA; Cary (NC): SAS Institute; 2006. Paper 096-31.
11 Maddox, JA, Fulenwider DO. Using the SAS system to analyze quality control data from data collection devices. In: Proceedings of the 15th SAS Users Group International; 1990 Apr 1–4; Nashville, TN, USA; Cary (NC): SAS Institute; 1990.
12 Kent P. Microcomputers and System 2000 DBMS. In: Proceedings of the 11th SAS Users Group International; 1986 Feb 9–11; Atlanta, GA, USA; Cary (NC): SAS Institute; 1986.
13 Hack-proof your passwords. Consumer Reports 2012; p. 19–21.
14 Wang X, Yu H. How to break MD5 and other hash functions. Lect Notes Comput Sci. 2005;**3494**:19–35.
15 Rivest RL. The MD5 message-digest algorithm. IETF RFC 1321. Fremont (CA): IETF; 1992.

Further Reading

1 Blair E. Using the SAS system for data collection under the VMS operating system. In: Proceedings of the 15th SAS Users Group International; 1990 Apr 1–4; Nashville, TN, USA; Cary (NC): SAS Institute; 1990. p. 888–895.
2 Jacobs CA. Installing and maintaining supervisor and product bundles in LPA/ELPA for MVS/XA and MVS/ESA sites. In: Proceedings of the 16th SAS Users Group International; 1991 Feb 17–20; New Orleans, LA, USA; Cary (NC): SAS Institute; 1991. p. 941–943.
3 Kahn D. The codebreakers: the story of secret writing. New York: Macmillan; 1967.
4 Kaminsky D. MD5 to be considered harmful someday. Cryptology ePrint Archive, Report 2004/357, 6 December 2004.
5 Klima V. Finding MD5 collisions — a toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 5 March 2005.
6 Kolb D. Micro to host link: performance and setup. In: Proceedings of the 12th SAS Users Group International; 1987 Feb 8–11; Dallas, TX, USA; Cary (NC): SAS Institute; 1987. p. 463–471.
7 Laing J. Client/server setup and implementation: web and non-web environments. In: Proceedings of the 25th SAS Users Group International; 2000 Apr 9–12; Indianapolis, IN, USA; Cary (NC): SAS Institute; 2000. Paper 19-25.
8 LeSueur J. Minimizing development risk while maximizing warehouse performance on UNIX® systems. In: Proceedings of the 25th SAS Users Group International; 2000 Apr 9–12; Indianapolis, IN, USA; Cary (NC): SAS Institute; 2000. Paper 120-25.
9 Mangold A. Regulatory compliant validation and deployment of SAS programs with the *Colibri* server. In: Proceedings of the Pharmaceutical Users Software Exchange; 2005 October 10–12; Heidelberg, Germany; Cary (NC): SAS Institute; 2005. Paper RC01.
10 Mironov I. Hash functions: theory, attacks, and applications. Redmond (WA): Microsoft Research; 2005 Nov. Technical Report No. TR-2005-187.
11 Shannon CE. The communication theory of secrecy systems. Bell Syst Tech J. 1949;28:656.
12 Singh S. The code book: the science of secrecy from ancient Egypt to quantum cryptography. New York: Anchor; 2000.
13 Schneier, B. Applied cryptography, protocols, algorithms and source code in C. New York: John Wiley & Sons, Inc.; 1996.
14 Wang XY, Guo FD, Lai XJ, Yu HB. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. rump session of Crypto'04. Cryptology ePrint Archive, Report 2004/199, 2004.