TYPE Original Research
PUBLISHED 30 November 2022
DOI 10.3389/ffutr.2022.914489

Integrity of virtual testing for crash protection

Esma Galijatovic[1,2], Maria Eichlseder[1]*, Simon Franz Heindl[2] and Corina Klug[2]*

[1]Institute of Applied Information Processing and Communications, Graz University of Technology, Graz, Austria, [2]Vehicle Safety Institute, Graz University of Technology, Graz, Austria

OPEN ACCESS

EDITED BY
Yong Han,
Xiamen University of Technology, China

REVIEWED BY
Hua-Lei Yin,
Nanjing University, China
Andre Eggers,
Federal Highway Research Institute, Germany

*CORRESPONDENCE
Maria Eichlseder,
maria.eichlseder@iaik.tugraz.at
Corina Klug,
corina.klug@tugraz.at

SPECIALTY SECTION
This article was submitted to Transport Safety, a section of the journal Frontiers in Future Transportation

RECEIVED 06 April 2022
ACCEPTED 09 November 2022
PUBLISHED 30 November 2022

CITATION
Galijatovic E, Eichlseder M, Heindl SF and Klug C (2022), Integrity of virtual testing for crash protection.
Front. Future Transp. 3:914489.
doi: 10.3389/ffutr.2022.914489

The interest in virtual testing is globally rapidly increasing because of several advantages compared to physical tests in laboratories. In the area of passive car safety, finite element simulations can be used to get further insights, use more biofidelic human models and make the overall assessment more robust by incorporating more variety in the virtual testing load cases. For a successful implementation of virtual testing in regulations or consumer information, the integrity of the procedure has to be ensured. As car simulation models used within the virtual testing are usually not shared with the evaluation institutions due to intellectual property (IP) issues, this is a challenging task. Stringent validation and certification procedures are needed and it has to be ensured that the models used in these steps are the same as the ones used for the virtual testing. In this paper, we developed a secure procedure for model version control. Through analysis of possible threats for both sides, car manufacturer and evaluation institution, we defined requirements, which the new procedure should satisfy. These requirements state that the integrity and authenticity of all shared documents should be protected, as well as the confidentiality of the simulation model. By considering all prerequisites, we developed an architecture for a new procedure. This architecture uses cryptographic algorithms such as hash functions and digital signatures to ensure integrity and authenticity, as well as secure computation mechanisms such as Intel Software Guard Extensions (SGX). In our proof-of-concept implementation, we demonstrated how a secure wrapper around LS-DYNA can produce a signed report to authenticate the input model files based on a hash tree and link them to the simulation results. The evaluation institution can use a matching verification tool to verify that the models were not manipulated compared to other simulation runs or the qualification process. The developed procedure can be used for trustworthy implementation of virtual testing into consumer information or regulation for the assessment of car safety with strengthened integrity. Further research is needed to develop comparable procedures for other simulation software packages or ideally integrate it directly into the simulation software.

KEYWORDS

car safety, consumer testing, integrity, confidentiality, hash function, digital signature, virtual testing

1 Introduction

The interest in virtual testing for vehicle safety assessments is globally rapidly increasing because of several advantages compared to physical tests in laboratories. Loading conditions can be varied as well as the anthropometry of the humans to enable a more robust assessment. Furthermore, integrated assessments linking pre- and in-crash phase and the use of biofidelic Human Body Models (HBMs) without hardware limitations are enabled. Thereby, instead of crashing a real car in a physical crashtests, a virtual replication of the car model is used and tested virtually in a simulated crash. (Huizinga et al., 2002; van Ratingen, 2020).

For a successful implementation of virtual testing, the integrity of the procedure has to be ensured. All involved parties and the end-consumers buying and driving the evaluated cars have to trust the procedure and potential manipulation must be prevented. As IP protected information is included in the virtual car models, this is a challenging task. Stringent procedures are needed to ensure quality of the model comparability in between such virtual tests (repeatability and reproducibility). The principle of such a procedure is shown in Figure 1.

The applied simulation models depend on the application case and the overall setup is often a combination of vehicle and occupant models. The models have to qualify for use in virtual testing by fulfilling requirements on the validation and/or comparability level. Thereby, general requirements (mass, geometry, output specifications..) are checked and model responses are compared with target responses from experiments and/or reference simulations. The assessment institution is inspecting shared results and documentation to certify the simulation models. Eventually, the qualified simulation models are used to run the virtual testing loadcases. The evaluation institution is inspecting the simulation results for plausibility (quality checks, consistency with qualification..) and considers the results of the virtual testing loadcases for the overall vehicle assessment. To replicate different loadcases, parts of the simulation models need to be changed "dynamic parts". Other parts of the model remain unchanged between qualification and virtual testing and also within the virtual testing procedure (named "static" the figure). It has to be ensured that the models used in the qualification steps are the same as the ones used for the virtual testing and cannot be manipulated in between. (Eggers et al., 2013; Klug et al., 2019; van Ratingen, 2020).

Currently, virtual testing procedures are still very rare in the area of car safety. In the European New Car Assessment Programme (Euro NCAP) (Euro NCAP 2022) pedestrian assessment of cars with deployable systems (i.e., pop-up bonnets), the first certification procedure for virtual human models to be used in the assessment procedure was defined. In this application case, the virtual tests—simulations where car models are crashing pedestrian models in different statures—are used to derive the boundary conditions for the physical tests. To qualify the virtual human models, they are used in reference simulations with generic car models and their response is compared to reference curves. Those simulations are very well-defined and as every user has access to the same generic car models, they are comparable among different users. This is done to qualify the simulation models and environments (cluster, used solver version) for use in virtual testing. Results and documentation of these reference simulations are shared with Euro NCAP and there inspected for plausibility. If Euro NCAP judges the all requirements are fulfilled and the reference simulations are within the corridors, the virtual human models qualify for the next step. There, the human models are impacted with the simulation models of the series-production cars to be evaluated. Both types of simulations are done by the car manufacturers, basically exchanging in the simulation setup only the generic vehicle models of the reference simulations with the models of the series-production cars for the assessment simulations. The consistency between the simulation setups and the virtual human models has to be documented and is checked based on the provided outputs. However, the integrity check purely relies on this documentation and the provided data and changes of the models are not trackable by third parties. In this case, this is accepted, as the simulations are only used as prerequisite for phsyical tests. For all simulations, quality criteria have to be fulfilled and plausibility of simulation results is inspected by Euro NCAP. (Klug et al., 2019).

Further applications of virtual testing are under development, where the simulation results should be used for the assessment itself and not only as prerequisite (Linder et al., 2020; van Ratingen, 2020). Therefore, the integrity of simulation results and used simulation models will play an steadily increasing role and further improvements might be needed.

In information security research and cryptology, data integrity is a central, well-studied security property. Cryptographic standards provide algorithmic solutions to protect the integrity of files, including hash functions (NIST FIPS 180-4, 2015), message authentication codes (MACs) (NIST FIPS PUB 198-1, 2008), and digital signatures (NIST FIPS PUB 186-4, 2013). These algorithms take as input a file and a suitable digital key (a symmetric key for MACs or the private key of an asymmetric keypair for signatures) and produce a fixed-size authentication tag, fingerprint, or signature to certify the integrity of the file. This can later be checked with the corresponding verification key (the same symmetric key for MACs or the public key of an asymmetric keypair for signatures) to verify that the file has not been modified. However, applying these generic algorithms to protect specific assets in practice is often challenging. The main difficulties include proper key management, secure implementation, and suitable adaptation based on the desired notion of integrity (e.g., if the asset includes parts that may be changed, if the asset is distributed across several locations or changes over time, taking metadata into account, and many other aspects).

The aim of our study was to explore possible solutions to improve integrity of the procedure and the files used in virtual testing by exploring methods from information security research and cryptology. The ambition was to define an enhanced procedure and investigate a proof-of-concept implementation.

2 Materials and methods

To investigate how the integrity of virtual testing could be further improved, the following methods were applied: First, we performed a threat analysis to structure potential problems, second, we developed and evaluated concepts to solve them, and third, we implemented a proof of concept in conjunction with a selected finite element (FE) software package, namely LS-DYNA (ANSYS - LST, 2021).

2.1 Threat analysis

The threat analysis was performed based on the current procedure for the assessment of active bonnets, having already future applications in mind, where the crash safety assessment itself is based on virtual load cases directly. Threats for the organisation performing the assessment as well as for the car manufacturer were considered based on discussions with different stakeholders.

2.2 Building blocks of the virtual testing procedure and the developed solutions

2.2.1 Simulations for virtual testing

For virtual testing in the field of vehicle safety, multibody or finite element simulations are performed. Different software packages are therefore used in the car industry. The main structure among the processes is very similar among the different Finite Element (FE) software packages. The current proof-of-concept implementation was done exemplary in combination with the FE software package LS-DYNA. The input files for LS-DYNA are plain text files that consist of ASCII-characters. All input files have to follow a certain format and structure. This includes a limited set of specific keywords, followed by respective values. By that, model elements and equations available in LS-DYNA are defined. Input files can also be encrypted, which means that keywords and related input parameters are only readable by LS-DYNA as it has the corresponding decryption key. When initiating the simulation, only one main input file is inserted into LS-DYNA, but many other files can be included by reference to achieve segregation of different components. These additional files are added using *INCLUDE and *INCLUDE PATH keywords. Outputs are binary files generated by LS-DYNA, which contain the simulation results (ANSYS LST, 2021).

An exemplary simulation setup for a simple "cube" example is shown in Figure 2. In this example, the files in the lower box remain unchanged (static parts of simulation model) throughout the procedure, while the main. key and Loadcase.inc. files are modified to change the loading conditions (dynamic parts of simulation model). This model is also used as demonstrator in the feasibility study.

As simulations for virtual testing are usually large models, simulations are performed on high-performance computing (HPC) clusters. These clusters are highly secured and connection to outside world is very restricted.

2.2.2 Cryptographic signature schemes and hash functions

Cryptographic or digital signing is a procedure that ensures the integrity and authenticity of a message (Diffie and Hellman, 1976; Rivest et al., 1978). This procedure is used when it is crucial to know who sent the specific message and ensure it was not altered in transit. Additionally, once a person signs a message with a digital signature, they can no longer repudiate their signature. This property is called non-repudiation. These procedures use asymmetric public-key cryptography with a key-pair of public and private keys, where the private key is used for signing by the sender, and the public key is used for verification by the receiver (Diffie and Hellman, 1976). Only a person that owns a private key can generate a signature, so digital signatures can not be forged. We have used two cryptographic signature schemes in this study: RSA (Rivest–Shamir–Adleman) and ECDSA (Elliptic Curve Digital Signature Algorithm), both standardized by US National Institute of Standards and Technology (NIST, 2013). The security of the RSA algorithm is based on the hardness of the integer factorization problem (Rivest et al., 1978), while ECDSA is analogue of the Digital Signature Algorithm (DSA) algorithm using elliptic curves (Miller, 1985; Koblitz, 1987). These schemes are considered secure against "classical adversaries", but would suffer a drastic loss of security in case of potential future "quantum adversaries" that have access to a large quantum computer. For this reason, NIST is currently searching for a "post-quantum"-safe replacement (NIST, 2020). Once such a post-quantum signature algorithm has been standardized, it can serve as a secure drop-in replacement for current algorithms provided that the increased cost of these algorithms (larger signatures, larger keys, slower computation) is compatible with the application. These are not to be confused with quantum signature algorithms (Gottesman and Chuang, 2001; Lu et al., 2021), which require that the system itself uses a quantum-bit public key and thus a dedicated quantum communication infrastructure. For this reason, they cannot serve as replacements in the context discussed in this paper.

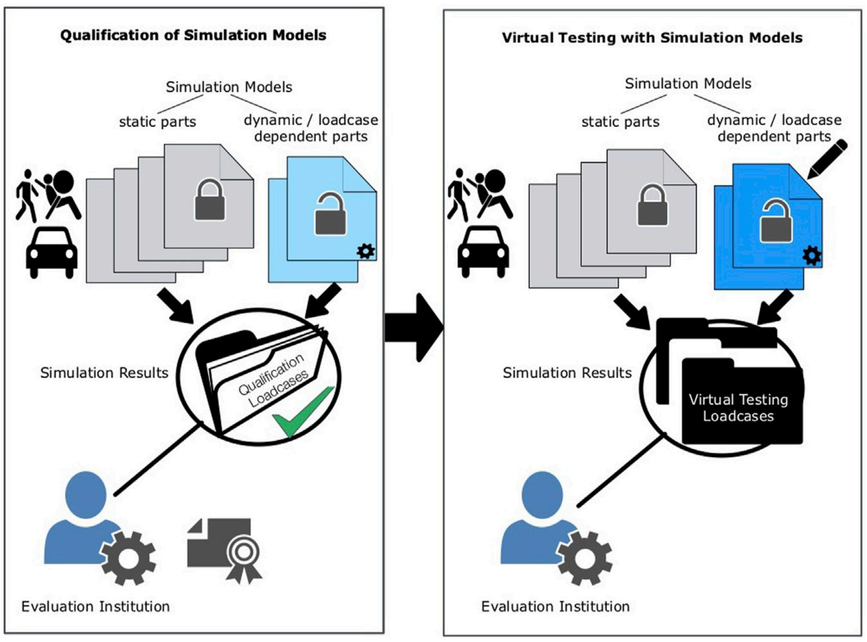

Qualification of Simulation Models
Simulation Models
static parts
dynamic / loadcase dependent parts
Simulation Results
Qualification Loadcases
Evaluation Institution
Virtual Testing with Simulation Models
Simulation Models
static parts
dynamic / loadcase dependent parts
Simulation Results
Virtual Testing Loadcases
Evaluation Institution

FIGURE 1
Possible implementation of virtual testing procedure where evaluation institution has access to simulation results.

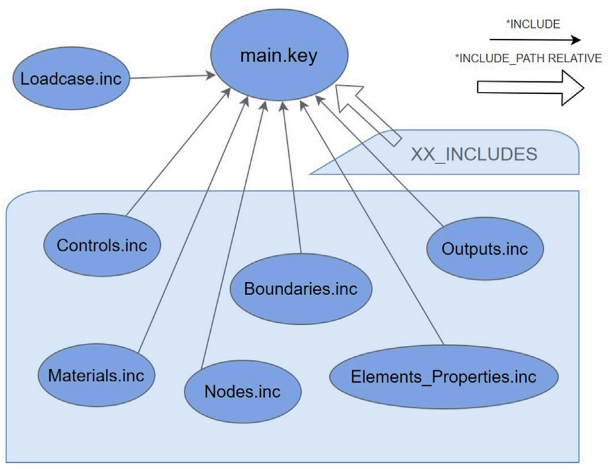


FIGURE 2
Simple “cube” example to illustrate the input file structure.

Additionally, all these cryptographic signature schemes rely on a cryptographic hash function to map input files of arbitrary length to a hash value, or tag, of fixed length that serves as a secure fingerprint (Damgård, 1989; Merkle, 1989). Secure cryptographic hash functions offer preimage resistance (i.e., it is infeasible for an attacker to find files that map to a given hash tag) and collision resistance (i.e., it is infeasible for an attacker to prepare two different files that map to the same hash tag). Hash functions and digital signatures are widely used as building blocks in cryptographic protocols (e.g., Transport Layer Security (TLS) for https or blockchains), file integrity (e.g., peer-to-peer (P2P) downloads), version control (e.g., git), and many other applications.

2.2.3 Secure enclaves

Secure enclaves (or Trusted Execution Environments) are secure subsystems of a computer with an aim to ensure confidential computing. Code executed in a secure enclave is protected from inspection or manipulation by other untrusted software, including higher privilege levels such as the operating system. This protects the confidentiality and integrity of the data processed by this trusted code. Thus, a program can run its most sensitive computations in a secure enclave; for example, the secure enclave can securely store cryptographic keys and allow their use only by the trusted cryptographic implementation and on this machine. The switch from unprotected, untrusted code to protected, trusted code in a secure enclave is implemented by a special interface (i.e., Intel’s call gate). One of the most prominent examples is Intel SGX (Intel, 2021) that can be used for key management, enhanced application and data protection, hardware-enhanced content protection, and more. They are entirely isolated from other processes, including the operating system. Intel SGX as a security mechanism is used in the developed procedure for private key management. This mechanism requires proper hardware support in terms of the secure enclave component inside of the processor.

3 Results

3.1 Threat analysis

The main threat for the evaluation institution is a mismatch of models used within the different steps of the procedure or a mismatch between simulation input files and outputs. These mismatches can either happen on purpose (e.g. optimised models for each loadcase separately) or as consequence of an error and could cause:

- A model is used by the car manufacturer for virtual testing that does not qualify for virtual testing as inconsistent model versions are used within the process.
- The simulation results shared have not been derived using the qualified models.

Leakage of confidential information is another possible threat. For car manufacturers, it is of great importance to keep simulation models confidential. Simulation model files contain protected intellectual properties, which is why they have to be protected against access from untrusted third parties. Simulation output binary files may also include confidential information since it may include information about the simulation model. Furthermore, the shared results also have to be protected against changes (by intention or accident).

Based on the threat analysis, we define the following requirements:

1) A unique identifier for a simulation model is needed to check consistency and protect integrity.
2) Some files are modifiable, others have to be consistent throughout all steps.
3) Some of the files contain confidential information of the car manufacturer or suppliers (e.g., material models), while load case-relevant information such as acceleration curves are not considered confidential. Confidential information has to be protected.
4) It has to be traceable from which simulation models the simulation results were computed.
5) The authenticity of the shared simulation results have to be ensured. Any modification of the exported results by any party must be detectable.

3.2 Architecture of the proposed approach

Based on these requirements, we developed a procedure and implemented a proof-of-concept.

3.2.1 Wrapper

A wrapper script was developed to read in the simulation files, run the simulation, extract the results, and calculate unique identifiers for each input and output file. As first step all simulation model files are read in, checked and hashed to generate the unique model identifier. Then the simulation is started using the cluster-specific shell script, which also specifies which simulation model is run. To avoid manipulation of the shell script, which is starting the simulation file in the time between hashing the input files and running the simulation, the wrapper is storing the content of the shell script internally and exports it into a new script file which is finally executed after the hashing was completed. As soon as the simulation terminates, the outputs of interest are extracted from the generated binout files into. csv files, which is then hashed too. Finally, all identifiers are listed in a report (i.e., a Portable Document Format (PDF) file) that needs to be signed. The process is shown in Figure 3.

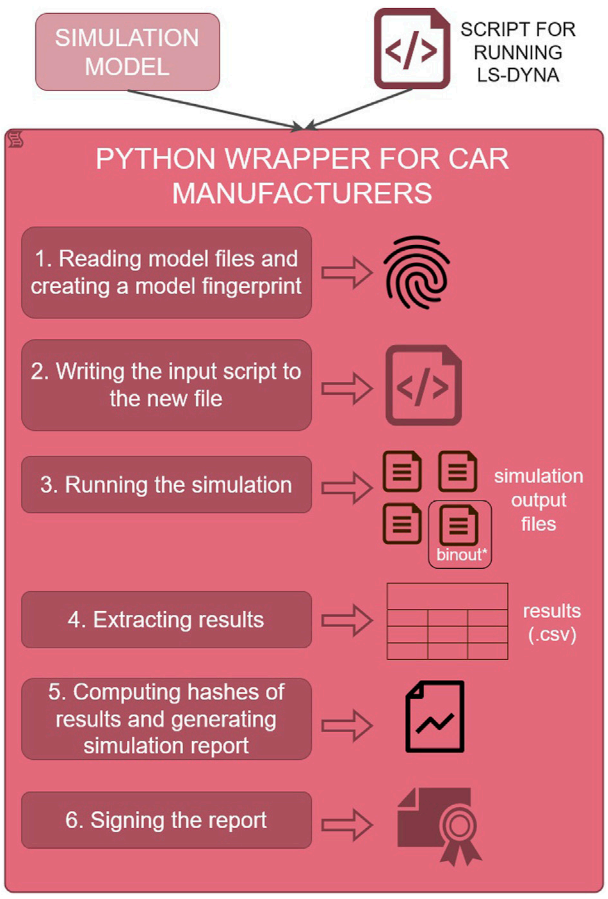

SIMULATION MODEL
SCRIPT FOR RUNNING LS-DYNA
PYTHON WRAPPER FOR CAR MANUFACTURERS
1. Reading model files and creating a model fingerprint
2. Writing the input script to the new file
3. Running the simulation
simulation output files
binout*
4. Extracting results
results (.csv)
5. Computing hashes of results and generating simulation report
6. Signing the report

FIGURE 3
Structure of the developed wrapper for use by the car manufacturer for running simulations, extracting results and signing simulation reports, including unique identifiers of inputs and results.

3.2.2 Read in the simulation files

Since the simulation model generally consists of multiple files cross-linked by INCLUDE keywords, we need a dedicated strategy to compute the hash. Any change in the file contents, file names, or file structure should produce a different hash. One simple approach is to concatenate the contents of all files and compute a model hash from this string. However, this design conflicts with the requirement of permitting a selected class of changes in some of the files as it would be not possible to distinguish between allowed and prohibited modifications. Therefore, we propose a different approach and represent the simulation model files as a tree structure, where one main file forms a root node and other files it includes form its child nodes. These child nodes can in turn contain other files. Knowing this, we can create a tree structure that captures all relevant information about the model and use it to compute the hash. Each node should contain information about file paths, file contents (hash of the contents), and files included (child nodes). An example of information kept in one node is:

This tree structure is convertible to a string that can be used for the computation of a SHA-256 hash, which is included in the final simulation report as the final model hash. With this method, it is clearly visible which files were changed within the procedure. Those files should not include any IP protected information and could be therefore shared with the evaluation institutions, while the IP protected information is clearly trackable with the unique identifier without access to the protected files by third parties. When this algorithm processes a single file, it first reads its content into a variable. Next, it looks for *INCLUDE PATH or *INCLUDE PATH RELATIVE statements, and if they exist, it adds these paths into the list of folders. The created list of folders contains all folders where the algorithm looks for child nodes. After, the algorithm searches for all *INCLUDE statements and processes the child nodes first. When all child nodes are added to the children list field (if there were any), the algorithm computes the SHA-256 hash of the contents and sets the corresponding field in the structure. This function then returns and continues the recursion until all files are visited.

Another important point is that the algorithm returns an error if a cycle is introduced. Assume, for example, that file X. key includes file Y. key, which includes file Z. key. If file Z. key includes file X. key, a cycle is introduced, and this is not an allowed situation. Therefore, the graph of files must be a tree (cycle-free). This situation is prevented by having a list of visited files forwarded through the recursion. This list contains hashes of all visited files. Then, when the file contents are read, and a hash is computed, it is first checked whether the hash of a current file exists in a visited files list. If not, the algorithm continues, and if this file was already visited, the algorithm raises an exception. Furthermore, to avoid that the user is overwriting parameters in the static files with keywords in the dynamic files, the inputs are also checked for duplicate parameters. If available, the input checker could in the future also look for forbidden keywords in the dynamic files.

After this object is created, it is translated into a string, and the final SHA-256 hash is computed from it. If any file is even slightly changed, this hash would be different. The tree structure here resembles tree structure showed in Figure 2. The only difference is that nodes contain additional information, as described previously.

3.3 Running the simulation

Setting up all needed parameters and variables for the simulation can be done using a shell script. This allows the person that is running the simulation to specify the cluster-specific settings (e.g. which LS-Dyna executable to use, License Server, Number of CPUs and the main simulation file which should be run). After the input script is processed, the simulation can be started. However, there is a time gap between reading the script and processing inputs and starting the simulation. This time gap could lead to a Time Of Check To Time Of Use (TOCTTOU) problem. Having such a problem could allow a malicious user to use one input script for preprocessing, then change the contents of the input script while the preprocessing is still under way and start the simulation with a completely different input script. To prevent this, we write the content of the input script to a new file with a random name and start the simulation using this newly generated file.

3.3.1 Simulation output

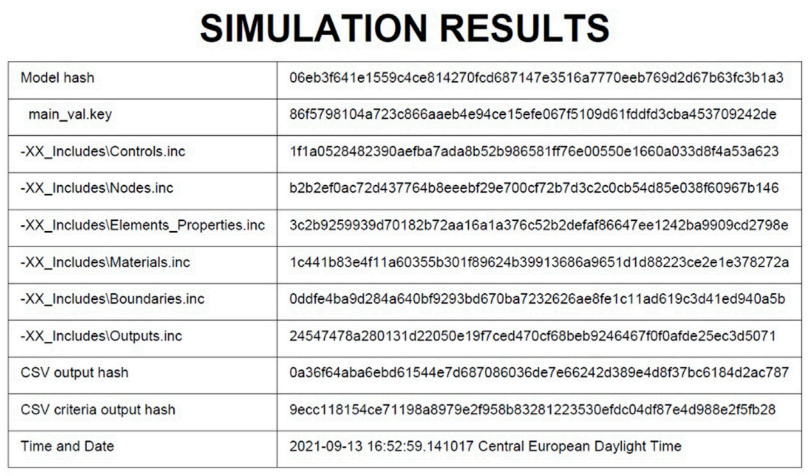
Typically, LS-DYNA outputs consist of several binary files, some of which contain confidential information about the model and data needed for the assessment of the car. The python libary "Dynasaur" (Klug et al., 2018; Schachner et al., 2022) was used to extract the outputs of interest from the binary output files and exports it into two CSV files (one including time series and one including scalars, such as injury criteria). These files can be shared with the evaluation institution, as they include the requested results used for plausibility checks and assessment (e.g. trajectories, acceleration signals, contact forces, calculated injury and quality criteria), but no IP protected information.

To track which model was used to produce these results and to ensure that results were not changed afterwards, another SHA-256 hash is calculated and included in the simulation report.

3.3.2 Simulation report

The simulation report includes the unique identifiers (hashes) of the simulation model and the simulation results. When all information about the model and results has been collected, a simulation report can be generated.

The report includes a table of hashes as illustrated in Figure 4 for the "cube" example of Figure 2. The final car model hash is presented in the first row, while relative file paths and hashes of their contents are included in the following rows. The final three rows contain CSV file hashes (two hashes computed from documents containing extracted results) and the time and



SIMULATION RESULTS

Model hash	06eb3f641e1559c4ce814270fcd687147e3516a7770eeb769d2d67b63fc3b1a3
main_val.key	86f5798104a723c866aaeb4e94ce15efe067f5109d61fddfd3cba453709242de
-XX_Includes\Controls.inc	1f1a0528482390aefba7ada8b52b986581ff76e00550e1660a033d8f4a53a623
-XX_Includes\Nodes.inc	b2b2ef0ac72d437764b8eeebf29e700cf72b7d3c2c0cb54d85e038f60967b146
-XX_Includes\Elements_Properties.inc	3c2b9259939d70182b72aa16a1a376c52b2defaf86647ee1242ba9909cd2798e
-XX_Includes\Materials.inc	1c441b83e4f11a60355b301f89624b39913686a9651d1d88223ce2e1e378272a
-XX_Includes\Boundaries.inc	0ddfe4ba9d284a640bf9293bd670ba7232626ae8fe1c11ad619c3d41ed940a5b
-XX_Includes\Outputs.inc	24547478a280131d22050e19f7ced470cf68beb9246467f0f0afde25ec3d5071
CSV output hash	0a36f64aba6ebd61544e7d687086036de7e66242d389e4d8f37bc6184d2ac787
CSV criteria output hash	9ecc118154ce71198a8979e2f958b83281223530efdc04df87e4d988e2f5fb28
Time and Date	2021-09-13 16:52:59.141017 Central European Daylight Time

FIGURE 4
Exemplary structure of the simulation report based on the “cube” example of Figure 2.

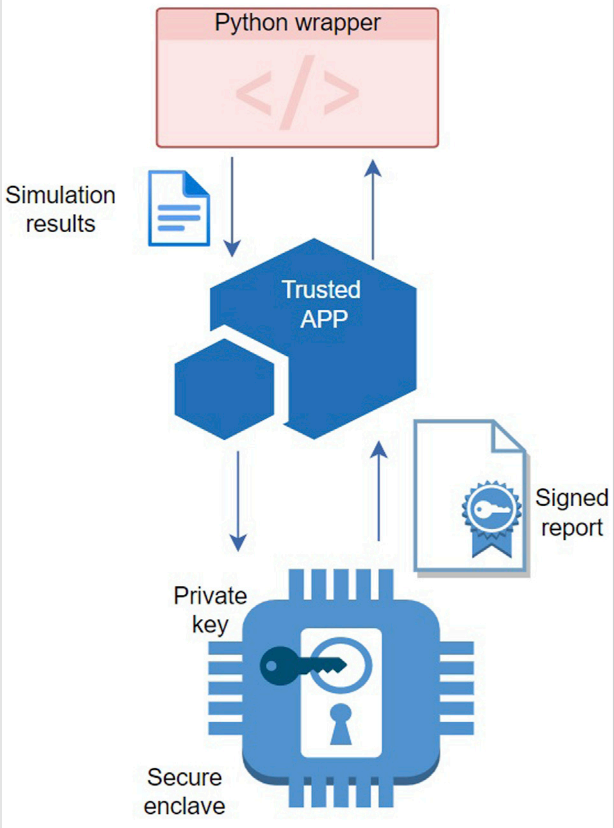


FIGURE 5
Communication between the Python wrapper, the trusted app, and the secure enclave which stores the private signing key.

date when the report is created. Invoking the build method over the document variable, a PDF document is generated and ready for signing. The report is saved to in a “Results”-folder along with the generated CSV files to be shared with the evaluation institution.

3.4 Signing approaches

When the model and results hashes are computed and included in the simulation report, it is necessary to prevent anyone from changing them, i.e., manipulating the report. This can be ensured by the evaluation institutions by signing the simulation report with a digital signature. Then, only the owner of the private signing key can create valid reports. However, as the report is created by the car manufacturer, managing this private key is challenging. In the following sections, we discuss several potential approaches.

3.4.1 Hiding the private key in a secure enclave

In this approach, the private key is created and stored in Intel SGX, as shown in Figure 5.

When the key is generated and stored inside the secure enclave, nobody can access it except the machine on which it was generated. Since the key would be generated for each car manufacturer separately, there would not be a unique key that belongs to the evaluation institution, but all of these signatures can be verified and used accordingly. A disadvantage of this approach is that there has to be hardware support. Car manufacturers must have secure enclaves in their processors on the HPC clusters or otherwise this approach cannot be used. The other disadvantage is that the car manufacturers could still use the private key for signing. Thus, it is necessary to bind the key to the process that generated it and thus limit its use, for example using remote attestation. That way, only the report generated by the script could be signed using that approach. The evaluation institution or a trusted third party hides the key inside the secure enclave of the car manufacturer's clusters and binds the key only to the script that runs the simulation.

To generate the signing key and sign the report using secure enclaves, in this solution, we use the Intel SGX application from Nevis (2020). It consists of two internal applications. The first one is the gateway application, which is an untrusted application that prepares the enclave and invokes its functions. The second one is the trusted application that runs in the Intel SGX enclave. It invokes the function *via* the Intel SGX ECALL mechanism.

The trusted app has two functionalities, key generation and data signing. In the key generation part, the trusted application generates a key pair for the ECDSA signature scheme, which consists of a private key and a public key. The private key is needed for signing and must be kept protected, while the public key can be exported to be used for verification of signatures. Technically, the private key is saved inside the enclave and encrypted using the Advanced Encryption Standard with Galois/Counter Mode (AES-GCM) algorithm that uses the encryption key derived from the silicon and the enclave's SIGNER measurement register. This way, the private key data is sealed and can only be unsealed in the same enclave on the same machine that created it.

The application can be used with simple commands to generate the public-private key pair and to sign the data. The key generation command generates a public key in Privacy-Enhanced Mail (PEM) format and stores it in the results directory. This file needs to be forwarded to the evaluation institution for signature verification. The private key and its associated information are stored in sealeddata. bin, protected by Intel SGX. The script first checks whether this file already exists in the root folder and generates the new key only if it does not. This prevents an attacker from causing denial-of-service by deleting the sealeddata. bin file. The command for signing needs the sealeddata. bin file to know which key to use for signing. In our proof-of-concept implementation, the resulting signature is stored in a Report. signature file next to the PDF report to be signed.

3.4.2 Signing by the person that runs the simulation

If secure enclaves are not available on the used hardware, signing could alternatively be performed by a person. This approach assumes a contact person who takes responsibility for model integrity. For the implementation in the proof-of-concept demonstration code, we use the "PDFNetPython" Python library to create a digital signature (PDFTron Systems Inc, 2001). To sign the PDF report using the PDFNetPython library, it is first necessary to create a signing widget in the document. This allows the user to create a field in a PDF document where the signature will be placed. The exact place of a field can be set using coordinates as parameters. There is also a possibility to add an image that will visually represent the signature. After the signature field is added, it is used to create a digital signature field object. This object is then saved to a variable, on which signing method is invoked using the private key file and the password which secures the private key file.

The signature algorithm used by this method is RSA with a key length of 2048 bit with SHA-256 hashing. The key file used as a parameter is in PKCS#12 file format (PFX filename extension) protected with a password. This file is exported using OpenSSL (The OpenSSL Project Authors, 2022). This approach improves the previous procedure significantly since it limits the time for potential model manipulation. However, model integrity could still potentially be subverted by a technically highly skilled adversary on the car manufacturer's side; this approach assumes a certain amount of trust between the responsible contact person and the evaluation institution.

3.5 Check of integrity

To prove that the procedure was performed properly, the evaluation institution needs to check if the received signed "Simulation Report" is consistent with the extracted simulation output files. Therefore, we developed a script as proof-of-concept implementation for the evaluation institution to perform the verification of the signature and the verification of the hashes (model hash and result hash). To check the integrity of the results, the testing institution computes the hash from the content of the result CSV file and compares it with the hash given in the report. If computed hash is the same, the integrity of CSV is preserved, since the hash given in the report is protected with a signature.

4 Discussion

4.1 Evaluation

4.1.1 Overview

In summary, our solution consists of several layers of protection which address different aspects of the problem:

- The hash function construction in the wrapper protects integrity of the simulation model: it records the details of the simulation model so that any differences can be detected. However: anyone could write down any hashes they like, i.e., this is not sufficient against a malicious actor, only against accidental errors.
- The signature protects the authenticity of the report: only the owner of the private key can produce a valid signed report and thus certifies the contents. This also provides non-repudiation, i.e., the owner of the private key cannot

deny having signed the report. However, access to this key must be restricted with organizational and/or technical measures. For example, it could be assigned to a trusted, personally responsible person.

- The trusted execution environment restricts the usage of the private key to a particular device and a particular trusted small program that implements the signing procedure. This prevents various kinds of misuse and implementation attacks. However, this signing procedure could still be invoked with malicious input (e.g., fake output of the simulation software by manipulating the interaction with this software; however, with the given setup, it would be very easy to trace who is responsible in case such a manipulation is suspected, and to check whether this was the case).
- A remote attestation setup (not implemented due to lack of network on target systems) could restrict who can invoke the trusted code and could monitor each invocation externally for further security if desired and feasible.

Next, we discuss the properties of this solution, as well as potential alternatives, in more detail.

4.1.2 Security

The developed procedure significantly improves the integrity and authenticity of related files by addressing the main threats identified. A cryptographic way to ensure the integrity of the simulation results was established. Unique model fingerprints are generated and documented to avoid that model files could be manipulated within the virtual testing procedure. Digital signatures guarantee the authenticity of a document and prevent repudiation.

4.1.3 Performance

The procedure brings only minor additional work for users and small computational overheads. To measure the real-time overhead, we run two processes over the mentioned cube example. One process is just the running the simulation, while the other process is started by running the implemented wrapper script (with the responsible person signing). The simulation (the baseline procedure without the wrapper) on the cube example lasted 13.0072 s, while the new procedure lasted 21.3379 s. When comparing these two times, it seems like there is a considerable time overhead (8.3307 s). However, taking into account that LS-DYNA simulations for real-world examples may often last even for several hours, the time overhead of 8.33 s is negligible and indistinguishable. The time overhead will be similar for a complex simulation since the same number of hashes, reports and CSV files are generated. Of course, if the complex solution has more files, the overhead will increase. However, it will still be within 1 minute, which is again negligible compared to a several-hour long simulation.

4.2 Limitations

While many mentioned threats are addressed, there are still some threats that one should be aware of. Since the simulation is not done in a protected environment, malicious users could still harm the procedure (e.g., by manipulation of the LS-DYNA executable), but a higher amount of criminal energy would be needed for that. Another threat is that the execution of the closed-source FE software itself can not be fully protected from manipulation; in particular, the signing procedure is currently not tied securely to the FE software (using remote attestation or similar mechanisms) and could thus be invoked separately by other processes on the same computer. The procedure with signing requires secure enclaves working on the computer hardware where simulations are run (e.g., HPC cluster at the car manufacturer). If this is not available, manual signing could be done, but this lowers the security level of the procedure.

The developed procedure requires that the files which can be modified do not contain IP protected information, so that the content can be inspected by the evaluation institution. Otherwise, additional contents and not allowed manipulations could be not avoided with the developed procedure. However, as load cases mostly differ in acceleration pulses and this is not supposed to be confidential information, our developed procedure could be used for such applications. If their are other use cases in the future where it would be needed to modify confidential parts of the model in between load cases another script checking the keywords in the changed files. This would require a list of allowed and forbidden keywords in the changeable files. Another approach could be Audits by trusted third parties underlying non-disclosure agreements. Our procedure would be still helpful for such audits as the information which files were changed in which loadcase is provided.

The procedure as such and the hash function also work on encrypted files, as the hash can be also calculated for the ciphertext to ensure its consistency between simulations. However, encrypted files cannot be audited and therefore parameters overwriting other parts of the models could be hidden in there. If parameters are defined multiple times, the developed wrapper could not identify it. Therefore, the current procedure can be not applied if the simulation setup consists of encrypted dynamic (load case dependent) parts. If static parts are encrypted, the procedure still works but is a bit weaker because duplicate parameters cannot be identified. The procedure was applied within this paper only for an easy example to make it more readable and easier to follow for the reader. Within the development phase it was also applied to bigger files with dummy models, which are closer to the final use case. It was found that the procedure works consistently, also with more complex files and file structures, basically changing only the result shown in Figure 4.

The procedure does not replace quality and plausibility checks of the simulations. This has to be done based on shared simulation results by the evaluation institution. The procedure just supports the evaluation institutions by documenting from which simulation models results were generated and which parts of the models were modified in between which steps of the procedure.

4.3 Alternative signing approaches

For report signing, it is necessary to have a private key known only by the evaluation institution or a trusted third party (e.g., LS-DYNA) and not by the car manufacturer. However, it is challenging to generate and safely store such a key. Several approaches were discussed within the development of the procedure described in this paper. The "secure enclave" approach was finally implemented and described within this paper. As the secure enclave approach still has the drawback that it might not be available on a certain hardware and the approach with a responsible person signing the report requires trust in this person, further alternatives should be investigated in the future, as we discuss next.

4.3.1 Using the private key from LS-DYNA

One option is to use LS-DYNA's private encryption key, which is intended for internal input file encryption and decryption in LS-DYNA. Using this key would be very convenient since it would solve both key generation and storage problems. Nevertheless, the level of provided security would depend on how well this key is protected. Only limited documentation on this embedded encryption method is publicly available, preventing alternative uses or security evaluation of this mechanism. Another option requring support from FE software developers is to hide the newly generated key inside the FE software binary.

4.3.2 Storing the private key on a server

Signing could also be done on a trusted third-party server. The implemented wrapper would have to create the final report and upload it to the server. Then, the server uses the stored key, signs the received document, stores corresponding details in a log file, and returns it to the wrapper.

This option has a significant shortcoming preventing its use given our requirements: The HPC clusters of car manufacturers typically do not have internet access or any connection to the outside world on purpose for security reasons, particularly due to the confidentiality of the files processed on these machines. Nonetheless, access to licence servers is needed also on these clusters. When the simulation is started, FE software could for example send the ID of the device to a licence server and receive "YES" if the machine has a valid license or "NO" otherwise. An idea based on this finding is to store the key on the very same machine as the licence server. This is, however, an unsuitable solution because of the possible responses from the server—it is not able to return a signed document but only a "YES/NO" answer. This would have to be changed to enable this option.

4.4 Future work

Additionally to the before discussed improvements for the signing, there are also other ways to further improve the procedure: Most importantly, the integration of the procedure in the FE software itself would make the procedure more easier to implement as the car manufacturer's clusters and further improve the security of the procedure because the wrapper functionalities would be embedded in the binary of the FE software. An input checker could check the keywords the files to avoid forbidden changes of the model especially when the dynamic parts of the simulation models cannot be inspected because of IP issues. Additional types of simulation results such as videos of animations could be also added to the procedure in the future, applying the same approaches, because basically every file can be hashed and the procedure therefore extended to everything where changes should be identified. Overall, our procedure is just addressing the integrity of a virtual testing procedure. Other future work has to focus on the development of the qualification criteria for the simulation models and the definition and assessment of the virtual testing loadcases.

5 Conclusion

In this paper, we propose a new and secure procedure for virtual testing of IP-protected simulation files. Security mechanisms, such as hash functions, digital signatures and secure enclaves are used to ensure the necessary security requirements. The proposed procedure makes it possible to trace which parts of the simulation model have been subject to modification without disclosing IP-protected information. Furthermore, the consistency between the simulation results and the input files can be checked. This ensures that models used in the previous qualification procedures was indeed used throughout the virtual test and that results were generated with the qualified models and therefore significantly improves the integrity of the procedure. We have implemented a proof-of-concept for the FE software package LS-DYNA and made it publicly available. Implementation for other simulation software packages (not necessarily restricted to FE software) is subject of future work. Further research is also needed to address known drawbacks of the proposed procedure (e.g. improve security of the signing process). The developed method is an enabler for increasing the integrity of virtual tests in consumer information or vehicle safety assessment regulations and could be implemented directly in FE software packages in the future.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: The code developed in this study is available *via* https://openvt.eu/Integrity_check/proof-of-concept-scripts.

Author contributions

EG wrote the first draft of the manuscript, performed the threat analysis and implemented the derived procedure in the sample code presented. ME wrote the parts of the manuscript focusing on security and supervised EG on the security aspects. CK wrote the virtual testing sections of the manuscript and supervised EG on the application aspects. ME and CK commented on the study. SH had the initial idea of such an approach and critically reviewed the manuscript. All authors contributed to the revision of the manuscript, read and approved the submitted version.

Funding

The master's thesis in which this study was performed has received support from the VIRTUAL project funded by the European Union Horizon 2020 Research and Innovation Program under Grant Agreement No. 768960.

Acknowledgments

The authors would like to thank colleagues from Dynamore and Scale for fruitful discussions of different possibilities for implementation.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

ANSYS - LST (2021). Ls-dyna. Available at: https://www.lstc.com/products/ls-dyna.

ANSYS-LST (2020). Keyword user's manual. *LS-DYNA*. https://www.dynasupport.com/manuals/ls-dyna-manuals/ls-dyna_manual_volume_i_r13.pdf

Damgård, I. (1989). A design principle for hash functions. *Adv. Cryptol.* 435, 416–427. doi:10.1007/0-387-34805-0_39

Diffie, W., and Hellman, M. E. (1976). New directions in cryptography. *IEEE Trans. Inf. Theory* 22, 644–654. doi:10.1109/TIT.1976.1055638

Eggers, A., Schwedhelm, H., Zander, O., Izquierdo, R. C., Polanco, J. A. G., Paralikas, J., et al. (2013). "Virtual testing based type approval procedures for the assessment of pedestrian protection developed within the eu-project imviter," in NHTSA, editor, The 23rd ESV Conference Proceedings (NHTSA).

Euro NCAP (2022). European new car assessment programme. Available at: https://www.euroncap.com/en.

Gottesman, D., and Chuang, I. (2001). Quantum digital signatures. doi:10.48550/arxiv.quant-ph/0105032

Huizinga, F., Ostaijen, R., and Slingeland, A. (2002). A practical approach to virtual testing in automotive engineering. *J. Eng. Des.* 13, 33–47. doi:10.1080/09544820110090304

Intel (2021). Intel software guard extensions. Available at: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

Klug, C., Luttenberger, P., Schachner, M., Micorek, J., Greimel, R., and Sinz, W. (2018). "Postprocessing of human body model results – introduction of the open source tool dynasaur," in CARHS, editor, 7th International Symposium: Human Modeling and Simulation in Automotive Engineering.

Klug, C., Feist, F., Schneider, B., Sinz, W., Ellway, J., and van Ratingen, M. (2019). "Development of a certification procedure for numerical pedestrian models," in NHTSA, editor, The 26th ESV Conference Proceedings (NHTSA).

Koblitz, N. (1987). Elliptic curve cryptosystems. *Math. Comput.* 48, 203–209. doi:10.1090/s0025-5718-1987-0866109-5

Linder, A., Davidse, R. J., Iraeus, J., John, J. D., Keller, A., Klug, C., et al. (2020). *VIRTUAL-a European approach to foster the uptake of virtual testing in vehicle safety assessment.* In 8th Transport Research Arena TRA 2020, April 27-30, 2020, Helsinki, Finland.

Lu, Y. S., Cao, X. Y., Weng, C. X., Gu, J., Xie, Y. M., Zhou, M. G., et al. (2021). Efficient quantum digital signatures without symmetrization step. *Opt. Express* 29, 10162–10171. doi:10.1364/OE.420667

Merkle, R. C. (1989). A certified digital signature. *Adv. Cryptol.* 435, 218–238. doi:10.1007/0-387-34805-0_21

Miller, V. S. (1985). Use of elliptic curves in cryptography. *Adv. Cryptol.* 218, 417–426. doi:10.1007/3-540-39799-X_31

Nevis, B. S. (2020). Gateway key provisioning and secure signing using intel® software guard extensions. Available at: https://software.intel.com/content/www/us/en/develop/articles/code-sample-gateway-key-provisioning-and-secure-signing-using-intel-software-guard.html.

NIST (2008). *FIPS PUB 198-1: The keyed-hash message authentication code (HMAC).* Gaithersburg, MD: National Institute of Standards and Technology: Federal Information Processing Standards Publication.

NIST (2013). *FIPS PUB 186-4: Digital signature standard (DSS)*. Gaithersburg, MD: National Institute of Standards and Technology: Federal Information Processing Standards Publication.

NIST (2015). *FIPS 180-4: Secure hash standard (SHS)*. Gaithersburg, MD: National Institute of Standards and Technology: Federal Information Processing Standards Publication.

NIST (2020). *Nistir 8309: Status report on the second round of the NIST post-quantum cryptography standardization process*. Gaithersburg, MD: National Institute of Standards and Technology Interagency or Internal Report.

PDFTron Systems Inc (2001). Digitally sign pdf files in python (2001-2021). Available at: https://www.pdftron.com/documentation/samples/py/DigitalSignaturesTest.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 120–126. doi:10.1145/359340.359342

Schachner, M., Micorek, J., Luttenberger, P., Greiml, R., Klug, C., and Rajinovic, S. (2022). Dynasaur - dynamic simulation analysis of numerical results. Available at: https://gitlab.com/VSI-TUGraz/Dynasaur.

The OpenSSL Project Authors (2022). The openssl project. Available at: https://www.openssl.org/.

van Ratingen, M.Update on virtual testing in safety assessments from euro ncap (2020). VIRTUAL OSCCAR workshop: Progress in Virtual Testing for automotive application. Available at: http://www.ircobi.org/wordpress/downloads/2020-virtual-osccar.pdf.