ExperItem Title	*collaborative Protection Profile for Full Drive Encryption – Authorization Acquisition*	**Reviewer**	*Paul Gallagher, NZ*
Item Identifier	*FDE-AA-cPP v0.2*	**Review Date**	1 December 2014
Version; Date:	*0.2; 2014-09-05*		

Notes :-

Severity	1	**Significant** - Impact the correct or efficient operation of the item. Needs discussion during a review meeting.
	2	**Moderate** - Normally clarifications or proposed improvements to the item which are unlikely to impact other areas. Probably doesn't need discussing at a review meeting.
	3	**Minor** - Does not affect the correct operation or interpretation of the item. These are usually syntax and format errors which have no effect on the meaning or interpretation of the item.

No.	Location	Comment	Suggested Change	Severity	Action

No.	Location	Comment	Suggested Change	Severity	Action
1.	Section 1.2	*Need for separate AA and EE documents.* *NOTED that Table 1 Page 10 does not identify a case where an AA or EE solution can stand in isolation: each requires a compatible other.* *NOTED that requirements on the API for discrete solutions is not defined, and is a "hard problem" to solve anyway.* *NOTED community already intends to provide guidance on evaluations using both elements together.* *NOTED that most text is repeated in each document: a combined cPP would not be substantially longer than either existing AA or EE document.*	Collapse the AA and EE documents to a single document. Even if a vendor only provides the AA or EE element, they must still operate with the other element in a provably secure way. Therefore it seems logical and efficient to require the complete solution be presented to any evaluation.	2	*Experience has shown it is problematic to require developers to come into evaluation with a partner to provide a complete solution. While it is true that both pieces are needed, one developer's product should not be held up in evaluation due to issues with another product. These products work with a number of different products, and should not be tied to whatever partner they can come into an evaluation with. The iTC believes that this is the best approach and allows flexibility for developers to manage the evaluation of their product. Therefore, the two cPP approach will remain.*

No.	Location	Comment	Suggested Change	Severity	Action
2.	Section 3.2: A.TRUSTED_CHANNEL	Disagree with the assertion that in a situation where independent products are used for AA and EE, that physical close proximity mitigates any threat that an actor may interpose itself in the channel between the two. An AA solution will almost certainly rely on a software-based actor on the host system. While I can "see" the bridge between an ID token and the host system, I will have no assured way of knowing if a malicious process is also operating on the host system, for example to tap information exchanges between the AA and EE. As we have not mandated any requirements on the API between the AA and EE, this represents a legitimate attack vector, and is not mitigated by proximity of components, as stated.	Either: - Collapse the AA and EE requirements to a single set, or: - Provide additional provable elements to the API to get the assurance we require. However, I believe this latter path may be too difficult to reasonably achieve.	1	*If there is a threat on the host system, then the data contained within the product must be assumed to be compromised, since the data is only encrypted on the TOE, whenever it reaches the host it is in the clear. No change to the cPP or SD.*

No.	Location	Comment	Suggested Change	Severity	Action
3.	Section 3.2: A.PLATFORM_STATE	This clause as stated actually allows a HDD encryption system to fail in the event malware hits the host PC, and still meet an "acceptable" level in terms of meeting this cPP. This is an arguably unrealistic "get out of gaol free' card. We need to run assured products in untrusted environments. I can never guarantee, for example, that a PC is malware free. What I instead need to do is provide guarantees that a security mechanism will remain trustworthy in all reasonable expected states of its environment. If I don't actually get that guarantee, I am looking at a worthless product.	Replace this requirement by one that requires the solution either operate correctly in all reasonably expected host system states (including potentially compromised) or as the only alternative, to fall into a fail-safe state.	1	*As with comment 2 above, if the host is compromised the TOE cannot be expected to provide any protections. FDE components do not provide anti-virus/anti-malware protection, so it is unreasonable for the FDE product to protect against all potential malware infections.*

No.	Location	Comment	Suggested Change	Severity	Action
4.	Section 3.2: A.SINGLE_USE_ET	Seems a user-unfriendly restriction, and in many cases probably a technically unnecessary restriction. Under this, a user who interacts with "n" solutions will have to carry a large pocket full of "n" tokens – neither popular nor desirable. Users get very creative when it comes to avoidance of this kind of setup and are more likely to do stupid things, like leave the tokens with/in the devices. As the information on the token has to be considered benign (i.e. like any public key split) there seems no real problem in allowing a single token to hold multiple public keys, and therefore if the system implementer has a mechanism to identify and use the correct key from within a collection, we should not prevent this happening.	Remove limitation: allow tokens to hold multiple credentials, but require that the stored credentials are benign to all systems except the target system.	2	*The use of tokens is not required. This assumption is in place to limit the proliferation of the tokens such that they may be gathered through another use, captured, and then used to compromise the data on the FDE – e.g., a user may use the token in another host (that may be not secured) and the token could be extracted. The iTC will discuss this change in the next version of the cPP.*
5.	Section 4.1: OE.SINGLE_USE_ET	Refer comments for A.SINGLE_USE_ET		2	*Please see above*
6.	Section 4.1: OE.PLATFORM_STATE	Refer comment above for A.PLATFORM_STATE			*Please see above*

No.	Location	Comment	Suggested Change	Severity	Action
7.	Section 5.1: Cryptographic Key Destruction	No failure notification. The requirement as stated record the key erasure processes, and the subsequent read-verification steps to confirm erasure. However in the (normally inevitable) event that the verification fails, while we intuitively know that this means the device has failed, there is no requirement on the system to actually react to this (the system could continue to operate in spite of the failure) nor to notify that the event has happened, nor any guidance on how the system should react to these events. NOTE ALSO: that while the comment is targeted at this one particular action, in general, any requirement to notify event failures is missing from the cPP.	- Add a requirement for a failure alert in the system API, - Require a fail-safe mode of operation from this point - Levy requirements on the management functions to provide active alerting in the event a failure condition has been detected. Also, review other sections for potential need for failure notifications.	1	*This is an area that will be explored further in the next version of the cPP.*
8.	Appendix E: Key Management	Since the key token forms a significant part of the solution, I would expect the vendor to also disclose a description of all physical media, the purpose and format of all elements stored on this media, and the mechanisms by which data is written and erased from the media.	- Expand requirements for the vendor's key management essay.	2	*The iTC will discuss this change in the next version of the cPP.*